SOUTHWEST RESEARCH INSTITUTE

6220 CULEBRA ROAD • POST OFFICE DRAWER 28510 • SAN ANTONIO, TEXAS, USA 78228-0510 • (512) 684-5111 • TELEX 244846

September 26, 1990

Mr. Larry D. Bishop
Building 30A, Room 2007B, DK331
NASA - Johnson Space Center
Houston, Texas 77058

Subject: Delivery of the Final Report for Research Into Alternative Network Approaches for Space Operations Support; NASA Grant NAG 9-369; SwRI Project No. 05-2921

Dear Mr. Bishop:

Enclosed with this letter is the final report pertaining to the research performed for alternative network approaches for space operations. This document includes the results of our research findings as well as the SwRI code that was developed for the proof-of-concept prototype effort. This delivery constitutes the completion of this research effort.

If you have any questions or comments, please call Dr. Antone L. Kusmanoff at (512) 522-5396.

Sincerely,

Melvin A. Schrader

Melvin A. Schrader
Director
Data Systems Department

MAS:ALK:pam

Enclosures (5 copies)

cc: Antone L. Kusmanoff [illegible]
Susan B. Crumrine [illegible]
Timothy J. Barton
William A. Bayliss
Noel C. Willis, Jr.
Thomas J. Purk, NASA-JSC, BG 211
✓NASA Scientific and Technical Information Facility (2 copies)

SAN ANTONIO, TEXAS
HOUSTON, TEXAS • DETROIT, MICHIGAN • WASHINGTON, DC

SOUTHWEST RESEARCH INSTITUTE
Post Office Drawer 28510, 6220 Culebra Road
San Antonio, Texas 78228-0510

RESEARCH INTO ALTERNATIVE NETWORK APPROACHES FOR SPACE OPERATIONS

FINAL REPORT

NASA Grant No. NAG 9-369
SwRI Project No. 05-2921

Prepared by:

Antone L. Kusmanoff, Ph.D.
Timothy J. Barton

Prepared for:

NASA
Johnson Space Center
Houston, Texas

September 27, 1990

Approved:

Melvin A. Schrader

Melvin A. Schrader, Director
Data Systems Department

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

1.0 INTRODUCTION

Information processing and associated computer-communication networks are constantly expanding to keep pace with the surging demands of an ever-increasing technological community. This is no more clearly seen than in one of the nation's hubs of technological application, the National Aeronautics and Space Administration's Mission Control Center (MCC) at the Johnson Space Center (JSC). The MCC serves as the nerve center for planning and controlling of manned space-flight missions. Over the years of its existence, the computer network architecture of the MCC has been iteratively redesigned due to technological advances and demands of the space operational team. The most recent step in this evolution is occurring during what is called the MCC Upgrade (MCCU). This kind of growth is happening not only at the JSC MCC, but at many other government and commercial locations that have heavily computerized support centers.

In the past, this kind of complex and somewhat rapid growth led to a situation where manufacturers responded by developing proprietary methods of computer-communication within their own product lines to provide the connectivity demanded by the spreading network. This maneuver meant that the computer-communication systems designers were required to integrate a variety of heterogeneous equipments, systems, and facilities into a smoothly functioning cohesive network to support their critical missions. What was really needed was a unified approach that could interconnect these dissimilar systems. Such a communications architecture would be "open" to all who comply with the rules of the architecture.

The Department of Defense (DOD), based on separate reports generated by the National Research Council (NRC) and the National Institute for Standards and Technology (NIST), has made a commitment to transition from their own communications architectures and protocol standards to the internationally recognized Open Systems Interconnection (OSI) architecture and OSI-related international standards. The general strategy that has emerged has three phases:

- o Phase I: Mandatory use of DOD military standards, normally referred to as transmission control protocol/internet protocol (TCP/IP), on procurements that involve networking and distributed processing.
- o Phase II: Adoption of corresponding international standards as co-standards to DOD standards.
- o Phase III: Exclusive use of international standards.

The NASA MCCU General Purpose Local Area Network (GPLAN) is following the Phase III procedure, exclusive use of the international standards, whenever possible. The difficulties that this phase brings about, and the reason for this research, is that users who are presently employing the DOD standard for their computer-communication needs must make software modifications to convert their current applications to operate within the international standards environment. This modification can be done quickly by some user applications, however others cannot make the transition by the time the network based on the CCITT/ISO standards is fully installed. Because of this, some users will lose their system functionality after the transition to international standards.

To allow progression to the newer standards, and still fully support its present users, NASA has instituted this research to determine if a method exists that will allow users of the DOD TCP/IP protocol family to send computer messages across a CCITT/ISO standards environment without modification to the original application software.

The answer to this research question begins with the analysis of a generic computer-communication background circumstance and finishes by considering the specific NASA JSC MCC implemented situation as a specific proof-of-concept demonstration. This report provides the strategy of conversion chosen which considers the parallel and divergent relationships within each of the protocol families considered. A detailed model of the strategy was derived and an implemented set of procedures was developed to prove out the solutions concept. The remainder of this introduction provides the background necessary to follow the research development activities.

1.1 Computer-Communications

The basic demand for communication presupposes the existence of the following: 1) at least two users, 2) information to be shared, 3) a common language between them, 4) a method to support the exchange, and 5) a method to control the exchange. All five of these elements are required for computer-communications. In this case, it is assumed that at least two user applications already exist, and that they are presently communicating information using the DOD TCP/IP as the method to control the exchange. The method of exchange, the fifth element above, is the one that is the center of this research study. Very little reference needs to be made to the details of the other four communication requisites as they already exist in a compatible state and will not change in relation to the research problem.

1.1.1 Standards

There are several bodies that are associated with publishing standards related to computer-communication. Some of the standards are as simple as defining common signaling speeds and others are as complex as describing the entire structure of a computer-communication architecture. The standard is a rule or set of rules which have been accepted by those concerned as a model to build a structure. Variation from the standards is not an uncommon event, and as can be expected, when enough vary from the current standard to a coinciding, but different new position, another standard is developed around that new position. A brief description of the major standards organizations is given below to eliminate confusion during the discussion in this area. Note that there are more standards organizations than those listed below, and these were chosen because of their dominate role in this study and the data communications industry.

1.1.1.1 <u>CCITT</u>

The International Telegraph and Telephone Consultative Committee (CCITT) is an operating arm of the International Telecommunication Union (ITU). The CCITT is an inter-governmental treaty organization (a United Nations treaty), and as such, its published standards are mandatory for use in Public Telephone and Telegraph organizations of a country. The interests of the United States are administered by the State Department, our voting member to CCITT. Since in the United States the telephone networks are privately owned, another level of membership to CCITT is also provided to recognize these private operating agencies. A membership level is also open to standards organizations such as the International Organization for Standardization (ISO). CCITT publishes its volumes every four years, currently the 1988 "blue books" use about four feet of shelf space covering a wide range of telecommunication concerns.

1.1.1.2 <u>ISO</u>

ISO is a voluntary organization founded to promote the development of international standards which will facilitate the exchange of goods and services and increase cooperation in technical and economic activities. ISO membership has approximately 70 member countries and 15 corresponding members. Membership is limited to the body most representative of standardization in a respective country. The United States is represented at ISO by the American National Standards Institute (ANSI). The activities of ISO are principally from the user committees and manufacturers in contrast to the carriers that are represented in CCITT.

1.1.1.3 <u>ANSI</u>

ANSI is a voluntary basis organization that was founded in 1918, although it was known by different names until its 1969 reorganization when it adopted ANSI. ANSI is a voluntary federation of approximately 200 organizations which represent commercial, trade, professional, consumer, etc. interests including government. ANSI finances itself with the sale of its standards and it dues. Technical committees recommend the standards and the Board of Standards Review approves the standards once it is assured that a consensus has been reached on the proposed standard. ANSI would normally adopt the ISO standards as it is a member of the ISO. However, if a specification differs due to unique aspects of North American systems, its specification will reflect that difference. The X3 committee was formed to establish standards related to computers and information processing and the OSI model.

1.1.1.4 <u>IEEE</u>

The Institute of Electrical and Electronic Engineers (IEEE) is a voluntary well-known professional society with chapters world-wide. It has most recently focused on the 802 structure for local area networks (LANs) providing standards that relate to the lower three levels of the OSI reference model. There are many other standards published by IEEE other than the 802 series; however, these are the only ones that provide interworking with the OSI reference model. Although the society has published the standards, the work building the standards was done by a subgroup within IEEE.

1.1.2 Standards Organization Relationships

The standards activities of most of the major organizations are closely interrelated. As was stated, ANSI is the United States member body in ISO. ANSI works with the State Department study groups to CCITT. ISO is a member of CCITT. There are other formal relationships between standards bodies, but just as important, there is also an informal infrastructure. Many times the same individual may hold two or more memberships, companies are represented in some manner in all of the major organizations, and delegations to international meetings are made up of the members of the different groups interested in the subject. These interrelationships are important to the process of standards development and the negotiation efforts that obviously occur. Their cooperation helps reach the widespread consensus necessary to install an international standard.

1.2 OSI Layering

In the OSI reference model, a system of seven layers are the foundation for the "open" nature of the protocol suite. According to this technique, each open system is viewed as logically composed of an ordered set of subsystems. Neighboring subsystems communicate through their common boundary by defined service requests and responses. There were several principles used as the basis to determine the number and nature of the layers. The CCITT X.200 and ISO 7498 standards both present these developmental principles for those interested. The two standards are nearly identical, however as this research deals with a United States governmental body, reference to terminology and definitions of CCITT X.200 will be followed when there is a conflict.

As a simple overview, the seven layers that are the basis of OSI communication will be briefly described with the primary reason(s) the layer is determined to exist. Note that every layer has associated with it both services and functions. In order for a layer to provide its services to the layer above it, the layer must execute those processes necessary to perform its functions. The functions are performed in concert with its peer layer throughout the network connections. The peer layers must accomplish the function by using the services of the layers below it. The peer protocols establish logical circuits (or virtual circuits (VCs)) between users by using the services of the lower layers. The following description for each of the layers will focus on the service the layer performs, and not on the details of the functions it must execute to perform its services.

1) The lowest layer in the model is called the Physical layer. This layer is responsible to connect, maintain and disconnect the physical circuit between the communication devices.

2) The Data Link layer has the responsibility of making the Physical layer appear error free to the network layer.

3) The Network layer provides network routing and switching through a network. The network consists of concatenated data links.

4) The Transport layer provides the first user-to-user level of peer protocols providing end-to-end accountability. This is the highest layer of the services associated with the providers of communication services. It does not matter how many links, nodes, or networks that are involved.

5) The Session layer serves as the organizer for the exchange of data between users. The session service binds together two cooperating user processes into a temporary communicating relationship.

6) The Presentation layer provides the syntax of the data in the model. It assures the application program properly interprets the data being transferred.

7) The Application layer is concerned with the support of an end-user application process. This layer is concerned with the semantics of the data. It supports application processes not a higher layer of the OSI model.

The OSI reference model provides a powerful tool for the communication designers use and implementation. However, it is only a baseline reference and not a specific solution. The partitioning allows planning as well as implementation and standards to proceed along clearly defined lines which can be combined into a cohesive solution even among multiple manufacturers.

1.3 NASA Environment

This research effort is being applied to a general problem that multitudes of networks are facing as they upgrade from TCP/IP to OSI based standards. The theoretical solution intent is to achieve a universal translation strategy that can be a pilot for specific network implementations. However, the verification process of this research includes developing code for a proof-of-concept activity that was generated against a specific network system. The network chosen supports the MCCU at NASA/JSC. It falls into the category of networks that have the specific multiple protocol problem, and its ready access to its internal workings were available for this investigation. This section very briefly discusses the network environment at JSC. Details can be found in the chapter on implementation for those who intend to use this research as an execution guide in a development of a translator system of their own.

As was stated earlier, the MCC serves as a nerve center for the planning and controlling activities associated with manned space-flight missions. In fact, it has done so since June of 1965 when the Gemini IV mission occurred. The custom built, proprietary, display and communication systems in the MCC with associated hardware drivers and interfaces were becoming obsolete. In addition, operators began to seek computational support outside of that directly provided by the MCC host systems. These trends led to the Mission Control Center Upgrade (MCCU) Program, which included mainframe computer replacement, the addition of supporting communication networks, and the addition of workstations as the foundation for the MCC monitor outputs. During this process it was also decided to follow the international communications standards where ever possible. An informative document released in 1987 titled "The Evolution of the Mission Control Center," spells out the history and the many features of the MCCU in sharp detail. It was written by Mr. Michael W. Kearney, an employee at NASA/JSC, and it can be found in the reference appendix to this document. The challenge of MCCU was large for NASA and its contractors, and is, in fact, still underway at the finish of this research project. The area of MCCU development that has greatest bearing on this research was the enhancement of the GPLAN which uses OSI based communication protocols as a replacement for the systems that previously used TCP/IP based message traffic.

2.0 INTEROPERABILITY STRATEGY

The goal of this research is to resolve the interoperability problem of applications employing the DOD TCP/IP family of protocols on a CCITT/ISO based network. The objective is to allow them to communicate over the CCITT/ISO protocol GPLAN network without modification to the user's application programs. There were two primary assumptions associated with the solution that was actually realized. The first is that the solution had to allow for future movement to the exclusive use of the CCITT/ISO standards. The second is that the solution had to be software transparent to the currently installed TCP/IP and CCITT/ISO user application programs.

2.1 Common Protocol

To continue to have application level interoperability it is necessary to have a common protocol (at the source and destination) from the user applications down to any crossover point that is developed. Within available implemented standards, it is theoretically possible in TCP/IP to approach a gateway function below IP as defined in TCP/IP documentation. With this method, for two systems to communicate with TCP/IP it is necessary for them to be compatible from above the gateway, the IP protocol, through the user application program. A solution of this kind would not be in line with the first assumption stated above. Also, it may not be a practical for processors in the system to have TCP/IP reside above the gateway system along with the required CCITT/ISO implementation protocols. Further, the installation of such a gateway would require a significant software development effort for the network support activities for users in such a dilemma.

Above the first four OSI communications layers, the higher level protocol layers are very diverse and it is much more difficult to engineer a method to convert between them. However, the international and DOD transport layer implementations have very common design features which was the result of a great deal of functional design compatibility associated with the OSI architecture and the TCP/IP in-place standard. Furthermore, in order to maintain the end-to-end transport operation feature, the transport layer virtual circuits must be preserved. Another consideration was that the transport layer normally resides within the operating system, tailored to the system it supports, meaning that the application program would not need modification. On the other hand, the upper three layers of the OSI model are usually found implemented within the application processes causing user application program modification mandatory, in violation to the second assumption.

These considerations, and others, drove the TCP/IP-OSI-TCP/IP Translator developed to be established at the Transport level of the OSI model. According to the CCITT and ISO international standards, the transport service is designed to provide the transparent transfer of data between session entities and relieves them from concern with the detailed way in which reliable and cost effective transfer of data is achieved. The protocols defined in the transport layer have end-to-end significance, where the ends are defined as correspondent transport entities. The transport layer is relieved of concern with routing, and relaying. The transport function invoked in the transport layer to provide a requested service quality depends on the quality of the network-service available. The negotiation of these and other factors is related to the transport level services. The general description of the services provided are as follows:

a. Establishes duplex transport-connections,
c. Maintains transport-connections to allow data transfer,
d. Releases transport-connections.

Each service provided by the transport layer can be tailored through the use of transport functions and facilities. The cooperation between the transport entities is controlled by the transport entities themselves.

The concern in this study is the actual interface services seen between the Transport Service User, hereafter called the TSU, and the transport protocol entity, hereafter called the Provider. In the DOD environment, the transport layer protocol is called Transmission Control Protocol (TCP) which is defined by Mil-Std-1778. When using the CCITT/ISO standard the transport layer is appropriately called the Transport Protocol (TP). CCITT/ISO goes a step further than TCP/IP by classifying five similar, but different, classes of transport level operation. The class that is applied at MCC is transport protocol class 4. For this reason, the transport protocol in the CCITT/ISO environment will be referred to simply as TP4.

Of course, below the transport layer is the network layer which appears as a provider to the transport entity. This relationship is transparent to the TSU, hence it can only recognize the transport entity as its Provider. In reality, all of the layers below the transport layer make up the function seen as the Provider to the TSU.

Within CCITT X.214, the interface definition to TP4, four transaction classes, called primitives, are invoked to and from the TSU through service access points (SAP) to the Provider. Within TCP the interaction primitives are grouped into two classes based on the direction of information flow. Downward directed communications are called service request primitives and upward directed communications are called service response primitives.

2.2 Single Terminology Basis

In this study, at times both of the standards are dealt with simultaneously. In an attempt to isolate their different terminologies and remain somewhat logical with the labeling, a new set of terms different from both standards will be used. With this language the primitives will be grouped in accordance to the direction the transactions occur across the Translator interfaces. If the direction of the primitive is from the TSU to the Translator or Translator to Provider it will be called a Petition Primitive. If the direction of the primitive is from the Provider to the Translator or Translator to TSU it will be called a Reaction Primitive. In accordance with this labeling, the below applies (this table is using the individual standards' languages):

a. Petition Primitives derived from the TP4 standard.

1)	TP4 Request:	Primitive by service user to invoke a function.
2)	TP4 Response:	Primitive by service user to complete a function previously invoked by an Indication at the SAP.

b. Reaction Primitives derived from the TP4 standard.

1)	TP4 Indication:	Primitive by service provider to invoke a function or indicate a function has been invoked.
2)	TP4 Confirm:	Primitive by service provider to complete a function previously invoked by a request at that SAP.

c. Petition Primitive derived from the TCP standard.

1)	TCP Request:	Primitive by TSU to enable connection establishment, data transfer, and connection termination.

d. Reaction Primitive derived from the TCP standard.

1)	TCP Response:	Primitive by Provider to inform TSU of various connection conditions.

2.3 Abstract Primitives vs Actual Code

In order to use a primitive, it must be coded into software with a very specific format. The primitive is issued to the layer to invoke the service entities and create headers that will be used by the peer layer in the remote station. Abstract primitive systems are used in the standards and in this study for the following reasons:

a. They permit a common convention to be used between layers.

b. They give the vendor a choice as to how the primitives will be implemented on a specific machine.

c. They ease the task of layer transportability between different vendor machines.

d. The use of standard primitives encourages the use of common formats of the data units.

The primitives provide a common base to compare transport services without taking a particular manufacturers software version or labeling procedures into account. In order for the design to be applied, the primitives eventually have to be integrated into implemented software. This occurred during the proof-of-concept phase for this NASA research (see Chapter 5) but in the following chapters, discussions will be using the general primitives as they are specified above.

After preliminary study, it appeared practical to translate TCP service petitions into TP4 primitives when two hosts, or users, are communicating with TCP/IP. Likewise, it appeared possible to translate reactions from the CCITT/ISO protocol suite as TCP primitives that were reactions to the TCP application. This translation is possible only where the TP4 protocol is a super set or common set of TCP. For features not available in TP4, the TCP/IP users must agree to not use those elements.

With this strategy, a processor station can maintain the TCP/IP applications while the network is installing CCITT/ISO communication software. Once the addition of CCITT/ISO upper layers is accomplished, the applications can begin to migrate to them until there are no longer TCP/IP application users. At that point, the translator can be extracted from the system.

The Translator will appear to a TCP/IP TSU as a standard TCP interface to the CCITT/ISO Provider. Likewise, the Translator will appear to the CCITT/ISO Provider as the TP4 interface to the TCP/IP TSU. In other words, the Translator will have one interface encountering the TCP/IP TCP TSU and the other interface encountering CCITT/ISO TP4 from the Provider. On the TSU-Translator interface the TCP Petition and Reaction Primitives will be received and issued. Similarly, on the Translator-Provider interface corresponding TP4 Petition and Reaction Primitives will be issued and received. Although not used in this research, it may be possible, and desireable, to define new local Translator to destination Translator management peer functions to assist in the management of the link and forward required information.

3.0 TRANSPORT LAYER STANDARD INTERFACE MODEL

This chapter discusses the transport layer interface service primitives as found in the DOD and the CCITT/ISO international standards. The service primitives defined for use on the interface to TCP will be discussed first. These primitives enable the needed functions that the application program applies for communication services. To have the need to use the Translator, it is assumed that these TCP service primitives have been implemented and are available through an application program between two users. The standard TP4 service primitives that exist to support the communications are discussed next in section 3.2. This section is a discussion of the transport layer service primitives that are presently available. Their actions must be known quantities to the Translator and are therefore explained here. The actions to be followed by the Translator to convert the TCP primitives into TP4 primitives will be provided in a later chapter.

3.1 Transmission Control Protocol Services to Upper Layer

The transport layer in TCP/IP called Transmission Control Protocol (TCP), MIL-STD-1778, was designed to provide reliable communication between pairs of processes in logically distinct hosts on networks and sets of interconnected networks. TCP is designed to operate successfully in an environment where the loss, damage, duplication, out of order data, and network congestion can occur. TCP appears in the DOD protocol hierarchy at the transport layer which is a counterpart with the transport layer defined in CCITT X.200 and IS 7082. TCP is defined to provide connection-oriented data transfer that is reliable, ordered, full-duplex, and flow controlled. TCP supports a wide range of upper layer protocols (ULP).

The major categories of service provided by TCP can be organized as follows:

a. Connection management services
b. Data transport services
c. Error reporting services

An interaction primitive defines the information exchange between two adjacent protocol layers. In TCP, information passed downward is called a "service request primitive." To stay within the language structure of the general framework established for this study in Chapter Two, service request primitives will be called Petition Primitives. In TCP, information units passed upward are called "service response primitives." Again, to align with the earlier defined structure, they will be referred to as Reaction Primitives. These interaction primitives need not occur in pairs. That is, a Reaction Primitive may occur independently of a Petition Primitive.

3.1.1 TCP Petition Primitives

The TCP Petition Primitives enable connection establishment, data transfer, and connection termination. The Petition Primitives are:

a. Unspecified Passive Open
b. Fully Specified Passive Open
c. Active Open
d. Active Open With Data
e. Send
f. Allocate
g. Close
h. Abort
i. Status

The Petition Primitive abbreviations, a description of the Petition Primitive action, and list of parameters for each Petition Primitive is given below. Optional Petition Primitive parameters are indicated by being followed by "[optional]."

3.1.1.1 Unspecified Passive Open (UPO)

This service Petition Primitive allows a ULP to listen for and respond to connection attempts from an unnamed ULP at a specified security and precedence level. TCP accepts in an Unspecified Passive Open at least the following information:

a. Source port
b. ULP timeout [optional]
c. ULP timeout_action [optional]
d. Precedence [optional]
e. Security_range [optional]

3.1.1.2 Fully Specified Passive Open (SPO)

This service Petition Primitive allows a ULP to listen for and respond to connection attempts from a fully named ULP at a particular security and precedence level. TCP accepts in a SPO at least the following information:

a. Source port
b. Destination port
c. Destination address
d. ULP timeout [optional]
e. ULP timeout_action [optional]
f. Precedence [optional]
g. Security_range [optional]

3.1.1.3 Active Open (ACO)

This service Petition Primitive allows a ULP to initiate a connection attempt to a named ULP at a particular security and precedence level. TCP accepts in an ACO at least the following information:

a. Source port
b. Destination port
c. Destination address
d. ULP timeout [optional]
e. ULP timeout_action [optional]
f. Precedence [optional]
g. Security [optional]

3.1.1.4 Active Open With Data (AOD)

This service Petition Primitive allows a ULP to initiate a connection attempt to a named ULP at a particular security and precedence level accompanied by the specified data. TCP accepts in an AOD at least the following information:

a. Source port
b. Destination port
c. Destination address
d. ULP timeout [optional]
e. ULP timeout_action [optional]
f. Precedence [optional]
g. Security [optional]
h. Data
i. Data length
j. PUSH flag
k. URGENT flag

3.1.1.5 Send (SND)

This service Petition Primitive causes data to be transferred across the named connection. TCP accepts in a SND at least the following information:

a. Local connection name
b. Data
c. Data length
d. PUSH flag
e. URGENT flag
f. ULP timeout [optional]
g. ULP timeout_action [optional]

3.1.1.6 Allocate (ALC)

This service Petition Primitive allows a ULP to issue TCP an incremental allocation for receive data. The parameter, data length, is defined in single octet units. This quantity is the additional number of octets which the receiving ULP is willing to accept. TCP accepts in an ALC at least the following information:

a. Local connection name
b. Data length

3.1.1.7 Close (CLS)

This service Petition Primitive allows a ULP to indicate that it has completed data transfer across the named connection. TCP accepts in a CLS at least the following information:

a. Local connection name

3.1.1.8 Abort (ABT)

This service Petition Primitive allows a ULP to indicate that the named connection is to be immediately terminated. TCP accepts in an ABT at least the following information:

a. Local connection name

3.1.1.9 Status (STT)

This service Petition Primitive allows a ULP to query for the current status of the named connection. TCP accepts in a STT at least the following information:

a. Local connection name

3.1.2 Service Reaction Primitives

TCP returns the Petitioned status information in a Status Reaction. Several service Reaction Primitives are provided to enable TCP to inform the user of connection status, data delivery, connection termination, and error conditions. The Reaction Primitives are:

a. Open Id
b. Open Failure
c. Open Success
c. Deliver
d. Closing
e. Terminate
f. Status Response
g. Error.

The Reaction Primitive abbreviations, a description of the Reaction Primitive action, and list of parameters for each Reaction Primitive is given below. Optional Reaction Primitive parameters are indicated by being followed by "[optional]."

3.1.2.1 Open Id (OID)

This service Reaction Primitive informs a ULP of the local connection name assigned by TCP to the connection requested in one of the previous service Petitions, UPO, SPO, or an ACO. TCP provides in an OID at least the following information:

a. Local connection name
b. Source port
c. Destination port [if known]
d. Destination address [if known]

3.1.2.2 Open Failure (OFA)

This service Reaction Primitive informs a ULP of the failure of an ACO Petition Primitive. TCP provides in an OFA at least the following information:

a. Local connection name

3.1.2.3 Open Success (OSC)

This service Reaction Primitive informs a ULP of the completion of one of the petitions that open service. TCP provides in an OSC at least the following information:

a. Local connection name

3.1.2.4 Deliver (DLV)

This service Reaction Primitive informs a ULP of the arrival of data across the named connection. TCP provides in a DLV at least the following information:

a. Local connection name
b. Data
c. Data length
d. URGENT flag

3.1.2.5 Closing (CLG)

This service Reaction Primitive informs a ULP that the peer ULP has issued a CLOSE service Petition. Also, TCP has delivered all data sent by the remote ULP. TCP provides in a CLG at least the following information:

a. Local connection name

3.1.2.6 Terminate (TRM)

This service Reaction Primitive informs a ULP that the named connection has been terminated and no longer exists. TCP generates this response as a result of a remote connection reset, service failure, and connection closing by the local ULP. TCP provides in a TRM at least the following information:

a. Local connection name
b. Description

3.1.2.7 Status Response (STR)

This service Reaction Primitive returns to a ULP the current status information associated with a connection named in a previous STT Petition Primitive. TCP provides in a STR at least the following information:

a. Local connection name
b. Source port
c. Source address
d. Destination port
e. Destination address
f. Connection state
g. Amount of data in octets willing to be accepted by the local TCP
h. Amount of data in octets allowed to send to the remote TCP
i. Amount of data in octets awaiting acknowledgment
j. Amount of data in octets pending receipt by the local ULP
k. Urgent state
l. Precedence
m. Security
n. ULP timeout

3.1.2.8 Error (ERR)

This service Reaction primitive informs a ULP of illegal Petition Primitives relating to the named connection or of errors relating to the environment. TCP provides in an ERR Reaction Primitive at least the following information:

a. Local connection name
b. Error description

3.1.3 TCP Service Sequenced Interchange Model

Using above service Petition and Reaction Primitives the functions of establishing, maintaining and releasing transport connections is achieved in a TCP/IP based network. The following sequences of transport level service primitives are the ordered procedures required to achieve these various activities on TCP/IP networks. The items below are required in the TCP/IP model, hence they will be the driving functions that have been translated and supported on the ISO network using the TP4 primitives.

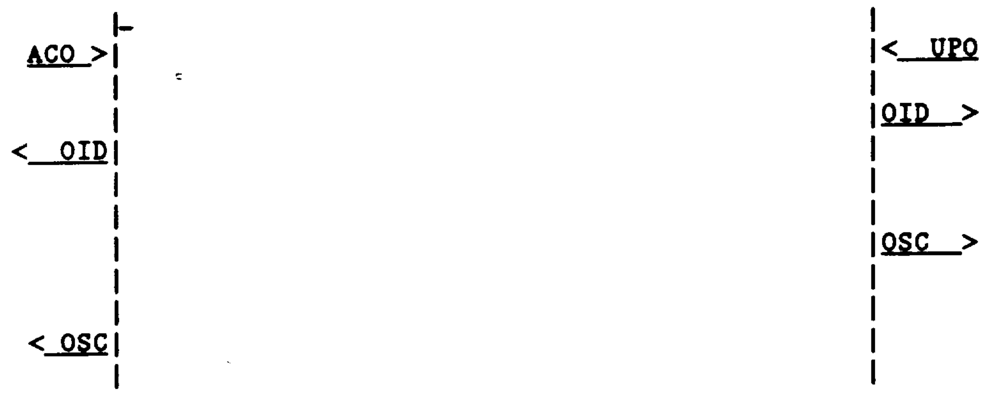

FIGURE 3-1. SUCCESSFUL TRANSPORT CONNECTION ESTABLISHMENT (PASS OPEN)

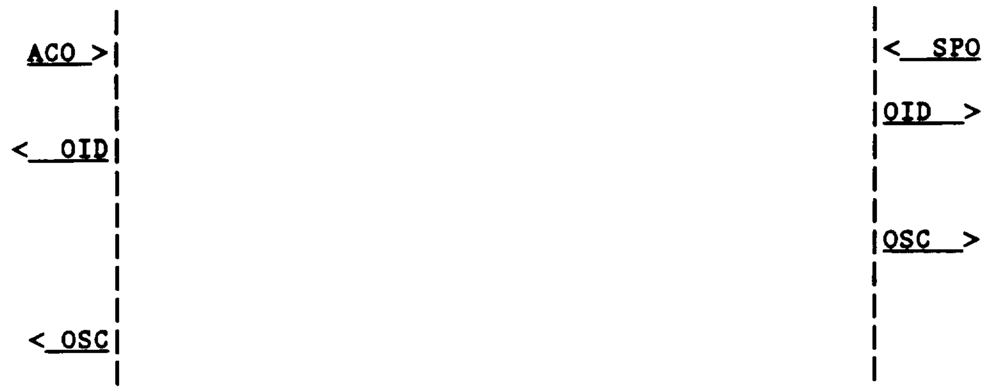

FIGURE 3-2. SUCCESSFUL TRANSPORT CONNECTION ESTABLISHMENT (SPEC OPEN)

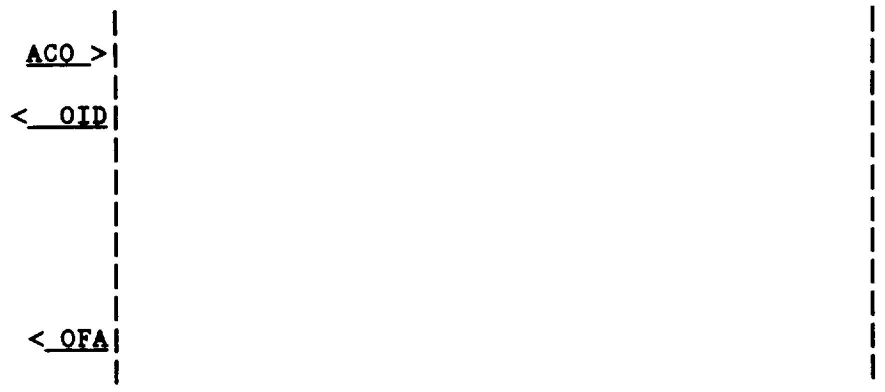

FIGURE 3-3. REJECTION OF TRANSPORT CONNECTION ESTABLISHMENT (NOT OPEN)

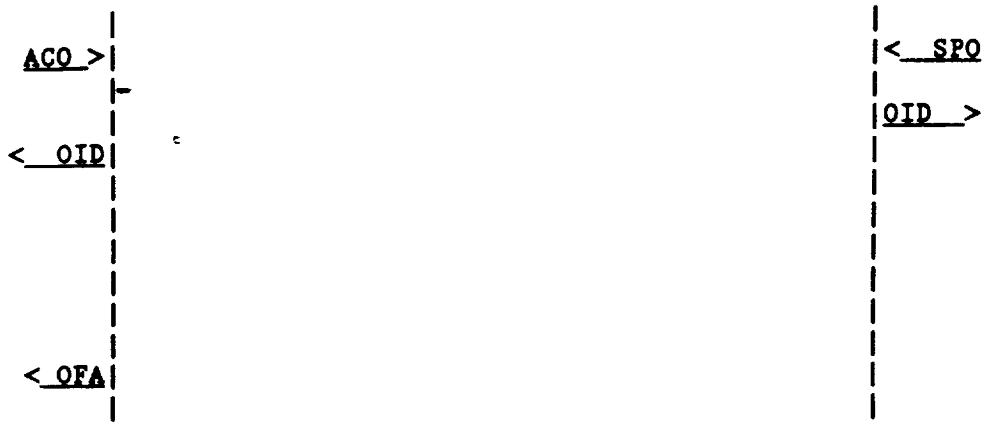

FIGURE 3-4. REJECTION OF TRANSPORT CONNECTION ESTABLISHMENT (PROVIDER)

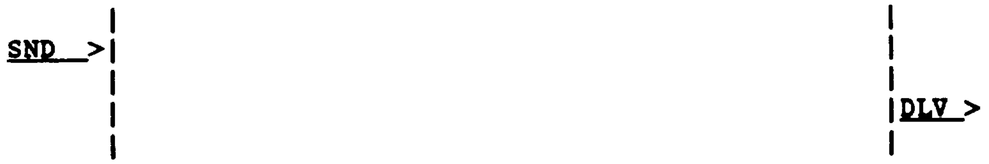

FIGURE 3-5. NORMAL DATA TRANSFER

FIGURE 3-6. TRANSPORT SERVICE PETITION TO PROVIDER

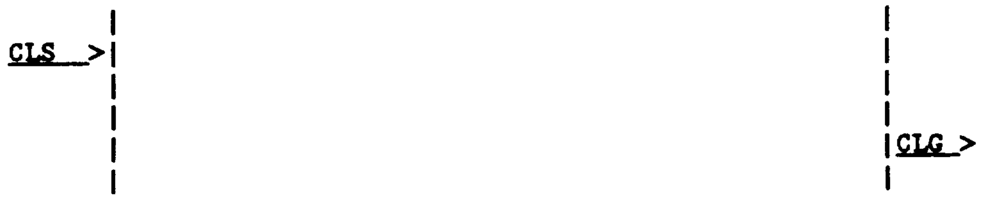

FIGURE 3-7. GRACEFUL CLOSE BY USER

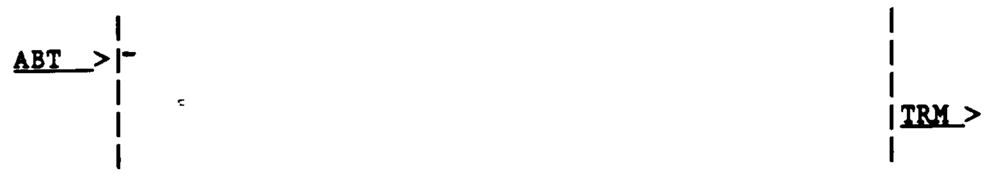

FIGURE 3-8. ABRUPT CLOSING BY USER

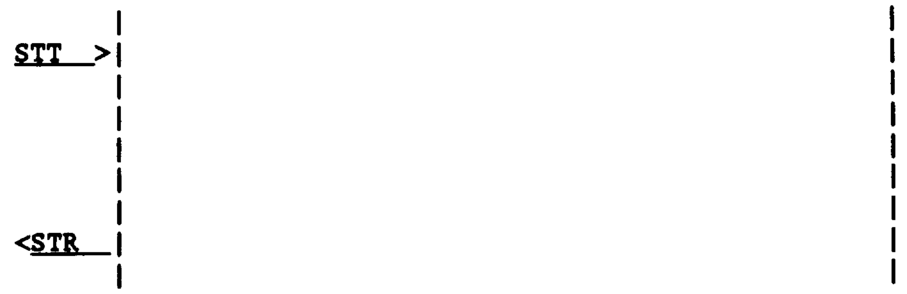

FIGURE 3-9. TRANSPORT SERVICE USER TO PROVIDER

FIGURE 3-10. PROVIDER TO TRANSPORT SERVICE USER

3.2 CCITT X.214/ISO IS 8072

Two international organizations, CCITT and ISO, have collaborated with their standards for the service methods to their transport layer based on the OSI reference model. The nearly identical documents, CCITT X.214 and ISO IS 8072, define the transport service interface that provides transparent transfer of data between Transport Service (TS) users on an OSI standards based network. Note that because the standards are essentially equal, reference will be limited to only the CCITT X.214 standard from this point on. X.214 is designed to relieve the TS users from concern about the detailed way in which supporting communications media are utilized to achieve this transfer.

The X.214 transport service provides for the following:

a) The means to establish a Transport Connection (TC) with another TS user for the purpose of exchanging Transport Service Data Units (TSDUs).

b) The means to provide a certain Quality of Service (QOS) as specified by the TS users.

c) The means of transferring TSDUs on a TC.

d) The unconditional release of a TC.

This list is compatible to the service features of TCP discussed in paragraph 3.1, except that the release available with OSI/CCITT transport service is only an unconditional "abrupt release" and does not define an orderly release. There are other differences that will be noted in detail later, however the Petition and Reaction Primitives that are available in this standard need to be presented before the detailed comparative analysis between TCP interface service features and the TP4 standard can begin.

3.2.1 CCITT Transport Level Service Petition Primitives

The TP4, X.214, Petition Primitives enable connection establishment, data transfer, and connection termination. The Petition Primitives are:

a. T-CONNECT Request
b. T-CONNECT Response
c. T-DATA Request
d. T-EXPEDITED-DATA Request
e. T-DISCONNECT Request

3.2.1.1 T-CONNECT Request (CNRQ)

The calling TSU invokes this function to initiate the establishment of a transport connection. The TSU specifies both the remote Transport Service Access Point (TSAP) and the remote Network Service Access Point (NSAP) address. The calling TSU can specify its desired QOS, and it can express its desire for the use of expedited data service. The following parameters are attached to this primitive:

a. Called address
b. Calling address
c. Expedited data [optional]
d. Quality of service [optional]
e. TS user-data [optional]

3.2.1.2 T-CONNECT Response (CNRS)

The called TSU invokes this function to accept an incoming connection. The called TSU may optionally transfer a limited amount of user data in the CNRS Petition Primitive. The following parameters are attached to this primitive:

a. Quality of service
b. Responding address
c. Expedited data [option]
d. TS user-data [option]

3.2.1.3 T-DATA Request (DARQ)

The TSU calls this function to initiate transmission of user data. The following parameter is attached to this primitive:

a. TS user-data

3.2.1.4 T-EXPEDITED-DATA Request (EDRQ)

The TSU calls this function to indicate to TP4 that it wishes to send expedited data on a TC. The connection must have been established with expedited data usage negotiated. The following parameter is attached to this primitive:

a. TS user-data

3.2.1.5 T-DISCONNECT Request (DSRQ)

The TSU calls this function either to disconnect an established Transport connection or to refuse an incoming connect indication. The following parameter is attached to this primitive:

a. TS user-data

3.2.2 Service Reaction Primitives

Several Reaction Primitives are provided to enable X.214 to inform the user of connection status, data delivery, connection termination. The Reaction Primitives are:

a. T-CONNECT Indication
b. T-CONNECT Confirm
c. T-DATA Indication
d. T-EXPEDITED-DATA Indication
e. T-DISCONNECT Indication

3.2.2.1 T-CONNECT Indication (CNIN)

The called TP4 layer invokes this function to communicate to the TSU a connect indication. TP4 specifies both the calling TSAP Selector Address and the calling NSAP Address. The following parameters are attached to this primitive:

a. Called address
b. Calling address
c. Expedited data [option]
d. Quality of service [option]
e. TS user-data [option]

3.2.2.2 <u>T-CONNECT Confirm (CNCF)</u>

The calling TP4 invokes this function to communicate to the TSU a connect confirm. The following parameters are attached to this primitive:

a. Quality of service
b. Responding address
c. Expedited data [option]
d. TS user-data [option]

3.2.2.3 <u>T-DATA Indication (DAIN)</u>

TP4 calls this function to notify the TSU that data has been received on the connection. The following parameter is attached to this primitive:

a. TS user-data

3.2.2.4 <u>T-EXPEDITED-DATA Indication (EDIN)</u>

TP4 calls this function to notify the TSU that expedited data has been received on the connection. The following parameter is attached to this primitive:

a. TS user-data

3.2.2.5 <u>T-DISCONNECT Indication (DSIN)</u>

TP4 invokes this function to communicate a disconnect indication to the TSU. The following parameters are attached to this primitive:

a. Disconnect reason
b. TS user-data [option]

3.2.3 <u>X.214 TP4 Service Sequenced Interchange Model</u>

As in the TCP based network, the above service Petition and Reaction Primitives are used in the functions of establishing, maintaining and releasing transport connections achieved in a TP4 based network. The following sequences of transport level service Primitives are the ordered procedures required to achieve these various activities on TP4 based standards networks. The items below are related to the same service items required in the TCP model, hence they will be the supporting structure that the translator must use.

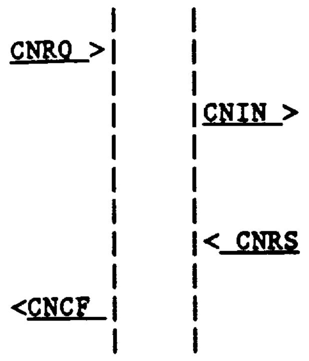

FIGURE 3-11. SUCCESSFUL TC ESTABLISHMENT

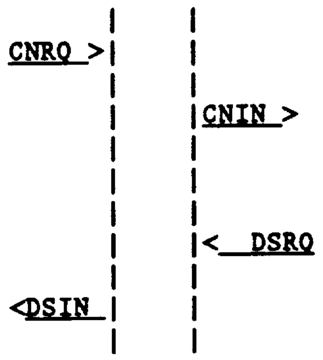

FIGURE 3-12. REJECTION OF TC ESTABLISHMENT (TS USER)

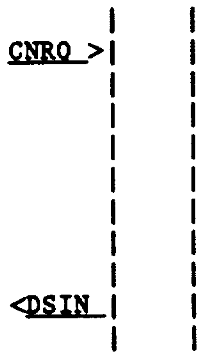

FIGURE 3-13. REJECTION OF TC ESTABLISHMENT (PROVIDER)

```
       |     |
DARQ  >|     |
       |     |
       |     |DAIN >
       |     |
```

FIGURE 3-14. NORMAL DATA TRANSFER

```
       |     |
EDRQ  >|     |
       |     |
       |     |EDIN >
       |     |
```

FIGURE 3-15. EXPEDITED DATA TRANSFER

```
       |     |
DSRQ  >|     |
       |     |
       |     |DSIN >
       |     |
```

FIGURE 3-16. TC RELEASE BY TSU

```
       |     |
       |     |
DSRQ  >|     |<DSRQ
       |     |
       |     |
```

FIGURE 3-17. TC RELEASE BY BOTH TSU PARTIES

```
       |     |
       |     |
<DSIN  |     |DSIN  >
       |     |
       |     |
```

FIGURE 3-18. TC RELEASE BY PROVIDER

```
       |     |
       |     |
DSRQ  >|     |DSIN  >
       |     |
       |     |
```

FIGURE 3-19. TC RELEASE INITIATED BY USER AND PROVIDER

4.0 TRANSLATOR INTERACTION

This chapter describes the interaction required by the Translator to fulfill its functions between the TCP and TP4 entities. The Translator is a programmatical function logically located between the TCP TSU, referred to only as the TSU from this point onward, and the TP4 Provider, referred to only as the Provider from this point onward. The Translator is able to operate within both the TCP and TP4 service interface languages, using the correctly associated Petition and Reaction Primitives. The following sections present each of the possible Petition Primitives from the TSU received by the Translator and the Petition Primitives that occur from the Translator to the Provider. The analysis that determines the correct conversion action necessary by the Translator is presented.

The first Translator function topic, section 4.1, will present the Translator and TSU interface relationship analysis. Then, section 4.2 follows with a Translator and Provider interface relationship discussion. The last discussion, section 4.3, presents the Translator sequence interchange model that must be complied with by the implemented code. This sequence is based on the individual service transport models developed in the earlier chapters. Once again, a possible confusion could have been caused if a preference to one of the standards' terminologies, "calling," "source" or "local" TSU, was applied within the Translator description for what is the originating TSU. Therefore, a different but equivalent label, "Origin TSU," was used within the Translator terminology when referring to the originating TSU. Likewise, the Translator terminology will refer to the "Target TSU" rather than the "called," "destination," or "remote" TSU as they are applied in the international standards.

4.1 Translator and TSU Interface Relationship Analysis

This section will address the various elements of each Petition and Reaction Primitive that is possible within a TCP conforming service and presents the proceedings necessary within the Translator to effect the conversion. The Translator will be using the TP4 Petition and Reaction Primitives to bring about the desired network operation over the TP4 based network. This section is a description of the events that occur both directions across the interface between the TSU and Translator based on initiation of interaction by the occurrence of a TSU TCP Petition Primitive. The TSU to Translator direction, the TSU TCP Petitions, will be presented first, followed by the Translator to TSU direction that is based on initiation of activity by the Translator TCP Reaction Primitives.

4.1.1 TCP TSU Petition Primitives

In the discussion of the Petition Primitives, the TCP Reaction Primitives will be applied when they are the required responses to the TSU by the Translator. The primitive will be described in terms of its standard, then the necessary Translator activity will be offered.

4.1.1.1 Unspecified Passive Open (UPO)

When an UPO Petition Primitive is issued by a local TSU it is informing the Provider that it is ready to receive, listening, from any other unnamed remote TSU. The local TSU provides its source port and optionally lists its timeout period, timeout action taken, its precedence allowed, and its security range.

TP4 service primitives do not support a generic passive listening status, so nothing will be petitioned to the Provider by the Translator as a result of the issuance of an UPO by the origin TSU. However, the Translator will be responsible to register the Origin TSU as being in an UPO status and listen to the Provider for any other Target TSU desiring to communicate. The optional data listed will need to be recorded into a status table for later application. The Origin TSU expects an OID Reaction Primitive which is the responsibility of the Translator to initiate and send.

4.1.1.2 Fully Specified Passive Open (SPO)

When a SPO is issued by a local TSU it is informing the Provider that it is ready to receive, listening, from a named remote TSU. The local TSU provides its source port, destination port, destination address and optionally lists its timeout period, timeout action taken, its precedence allowed, and its security range.

TP4 service primitives do not support a similar passive specific listening status, so nothing will be petitioned to the Provider by the Translator as a result of the SPO. The Translator will be responsible to register the Origin TSU as being in an SPO status and listen to the Provider for the named Target TSU desiring to communicate. The optional data listed will need to be recorded into a status table for later use. The Origin TSU expects an OID Reaction Primitive which is the responsibility of the Translator to initiate and send.

4.1.1.3 Active Open (ACO)

When an ACO is issued by a TSU it is informing the Provider that it desires to initiate a connection with a named remote TSU. The local TSU provides its source port, destination port, destination address and optionally lists its timeout period, timeout action taken, its precedence allowed, and its security range just as in the SPO. The difference with this open primitive is that it is asking if the named Target TSU has issued a UPO, a SPO or an ACO with it as the destination.

TP4 service primitives support a named active open status, so a CNRQ will be petitioned to the Provider by the Translator as a result of the ACO. The Translator will be responsible to register the TSU as being in an ACO status and issue the CNRQ to the Provider for the Target TSU. The optional data listed will need to be recorded into a status table. The Origin TSU first expects an OID Reaction Primitive which is the responsibility of the Translator to initiate. Then, it expects to receive either an OFA or an OSC depending on the status of the named Target TSU.

The Translator will need to issue a CNRQ to the Provider providing the called address, calling address, and provide a reset optional expedited data flag, list the QOS desired, and not provide any user-data. The QOS parameter must be either default or modified depending on the requesting TSU optional data in the ACO. There will need to be an address mapping accomplished to convert the TCP/IP addresses into TP4 addresses.

4.1.1.4 Active Open With Data (AOD)

When an AOD is issued by a TSU, it is informing the Provider that it desires to initiate a connection with a named remote TSU and has data that should be sent along with the request. The origin TSU provides its source port, destination port, destination address and optionally lists its timeout period, timeout action taken, its precedence allowed, and its security range. The difference between this primitive and the ACO is that the AOD is asking if the named Target TSU has issued a UPO, a SPO or an ACO with it as the destination and has user-data to be transmitted during the request.

TP4 service Primitives support a similar named active open status with data, so a CNRQ will be petitioned to the Provider by the Translator as a result of the AOD. The Translator will be responsible to register the TSU as being in an AOD status and issue the CNRQ to the Provider for the Target TSU. The optional data listed will need to be recorded into a status table. The Origin TSU first expects an OID Reaction Primitive which is the responsibility of the Translator to initiate. Then it expects to receive either an OFA or an OSC, depending on the status of the named Target TSU.

The Translator will issue a CNRQ to the Provider providing the called address, calling address, and provide a reset optional expedited data flag, list the QOS desired, and provide the user-data. The QOS parameter must be either default or modified depending on the requesting TSU optional data in the AOD. There will need to be an address mapping accomplished to convert the TCP/IP addresses into TP4 addresses.

4.1.1.5 Send (SND)

This petition is the Petition Primitive that causes user-data to be transferred across an already named connection. The SND will provide the local connection name, the user-data it wishes to send and the length of the user-data. Two flags, the PUSH and URGENT flags can be set or reset. The PUSH flag indicates to the Provider to not wait for any more data to fill a buffer, but to transmit all user-data up to and including this block now. The URGENT flag is for the destination to be aware of the fact that the sending TSU has marked this data as urgent. This would relate to a prior agreed upon signal between the two TSU's. Also optionally included is the timeout window and action requested by the TSU in case of timeout.

The Translator will issue a DARQ to transmit the user-data to the Provider, or eventually the Target TSU. The Translator will not need to react to the PUSH option because the TP4 protocol sends out data grouped into octet blocks. The basic service Primitive DARQ does not include a block length.

4.1.1.6 Allocate (ALC)

The ALC petition is an explicit flow control procedure that reports to the Provider the amount of data in octet units that it is willing to accept. It is an incremental count.

The Translator using TP4 has a similar explicit flow control by defining the Quality Of Service (QOS) for the TC as observed between the endpoints. There is a loosely defined, multifunctional set of parameters that are used during TC establishment. A combination of message buffer storage by the Translator (unacknowledged, however) and QOS selection would be necessary to supply the TSU with the correct reaction to the ALC Petition Primitive.

4.1.1.7 Close (CLS)

With the CLS Petition Primitive the TSU is reporting to the Provider that it has completed the user-data transfers across the TC and wishes to terminate the TC. With a CLS, the termination is not immediate. If there is message traffic from the Target TSU then the TC is maintained so that it is allowed to complete the current message before terminating the TC.

TP4 does not allow for a similar event, in that it can only immediately issue its DSRQ and the TC is abruptly terminated. To maintain the graceful closing procedure as in CLS it is necessary for the Translator to delay sending the terminate (TRM) until the completion of the current receive message (if one exists). Upon receipt by the TSU of the current message, a DSRQ can be issued to the Provider to terminate the TC.

4.1.1.8 Abort (ABT)

The issuing of an ABT by the TSU is a request which can not be denied and it has an effect to immediately terminate the TC. Data may be lost in transition.

The Translator will advance a DSRQ without delay to the Target TSU which will terminate the TC immediately.

4.1.1.9 Status (STT)

This petition allows the TSU to obtain status of the named connection.

The Translator is required to issue a STR to the origin TSU. The information attached to the STR will have been maintained locally by the Translator.

4.1.2 TCP Translator Reaction Primitives

In the discussion of the TCP Petition Primitives, the Reaction Primitives were presented as they were the required responses for the Translator. This discussion will repeat some of that information in this context, but will also be presenting the conditions necessary for the Translator to issue the service Reaction Primitive listed.

4.1.2.1 Open ID (OID)

Through this Reaction Primitive, TCP normally informs the TSU of the local connection name assigned by TCP to the connection requested in one of the previous service Petitions, USO, SPO, ACO or an AOD. TCP is required to provide at least the following information:

a. Local connection name
b. Source port
c. Destination port [if known]
d. Destination address [if known]

Upon receipt of the USO, SPO, ACO, or AOD petitions the Translator provides to the TSU the local connection name for the TC requested. In the cases of USO and SPO, registration of the TSU under the local connection name assigned occurs, and a change to the listening state by the Translator follows. In the case of ACO and AOD the Translator will be required to attempt to open a connection to the Target TSU. This is accomplished through the Translator by issuing a CNRQ to the Provider using the correctly mapped TP4 address to the Target TSU.

4.1.2.2 Open Failure (OFA)

This service Reaction Primitive is sent to inform a TSU of the failure of an ACO or ACO service Petition Primitive. TCP normally would provide in an OFA at least the local connection name.

In response to ACO or AOD Primitive the Translator must provide to the TSU an OFA Reaction Primitive to indicate that the Provider was unable to provide a TC to the Target TSU. The Translator will have received a DSIN from the Provider as a reaction to an earlier CNRQ petition in order to issue the OFA.

4.1.2.3 Open Success (OSC)

This service Reaction Primitive is sent to inform a TSU of the completion of one of the open service petitions. TCP would normally provide in an OSC at least the local connection name.

In response to USO, SPO, ACO or AOD Petition Primitive the Translator must provide to the TSU an OSC Reaction Primitive to indicate that the Provider was able to provide a TC to the Target TSU. The Translator will have received a CNCF from the Provider as a reaction to an earlier CNRQ petition in order to issue the OSC.

4.1.2.4 Deliver (DLV)

Using this service Reaction Primitive, TCP informs a TSU of the arrival of data across the named connection. TCP would normally provide in a DLV at least the following information:

a. Local connection name
b. Data
c. Data length
d. URGENT flag

When the Translator receives user-data from the Provider through a DAIN it forwards the data to the TSU with a DLV Reaction Primitive. In the TP4 Primitives established, there is no way for the Translator to be aware of the setting of the Urgent flag or the data length transmitted by the sending TSU. The URGENT flag and data length would be lost without a Translator to Translator mechanism to report the request. The URGENT flag is a higher level indication, and no action is taken by TCP when it is present, therefore it can be ignored without impact in the communications environment. The Data length value can be regenerated by the Translator, so it does not need to be communicated across the network.

4.1.2.5 Closing (CLG)

TCP uses the CLG Reaction Primitive to inform a TSU that the remote TSU has issued a CLS service Petition. Also, TCP has delivered all data sent by the Target TSU. TCP provides in a CLG, at least the local connection name.

The CLG will never be sent to the TSU from the Translator because TP4 does not support the CLS Reaction Primitive.

4.1.2.6 Terminate (TRM)

This service Reaction Primitive informs the TSU that the named connection has been terminated and no longer exists. TCP normally generates this response as a result of a remote connection reset, service failure, and connection closing by the local ULP. TCP provides in a TRM at least the local connection name and description of why the TRM was issued.

The TRM will be issued to the TSU from the Translator whenever it receives a DSIN from the Provider.

4.1.2.7 Status Response (STR)

The STR Reaction Primitive returns to a TSU the current status information associated with a connection named. TCP normally is required to issue a STR to the TSU providing the following information to the TSU:

a. Local connection name
b. Source port
c. Source address

d. Destination port
e. Destination address
f. Connection state
g. Amount of data in octets willing to be accepted by the local TCP
h. Amount of data in octets allowed to send to the remote TCP
i. Amount of data in octets awaiting acknowledgment
j. Amount of data in octets pending receipt by the local ULP
k. Urgent state
l. Precedence
m. Security
n. ULP timeout

All of the listed items except item i. are available to the Translator from TP4 entity or previous TSU interaction using the Primitives and stored table data available. The value of item i. will need to be tracked by the Translator in its interaction with the Provider.

4.1.2.8 Error (ERR)

This service Reaction Primitive informs the TSU of illegal service petitions relating to the named connection or of errors relating to the environment. TCP uses this Primitive to provide an error reaction to the TSU Reaction Primitive providing at least the local connection name and a method of providing the error description.

The Translator is responsible for generating and submitting to the TSU the ERR Reaction Primitive.

4.2 Translator and Provider Relationship Analysis

This chapter discusses the various elements of each Petition Primitive and Reaction Primitive that exists in a conforming TP4 service interface set. The section also presents the proceedings necessary within the Translator to accomplish the primitives identified. The Translator uses TCP service primitives to bring about the desired operation with the TCP TSU. This section describes the events that occur both ways across the interface between the Translator and Provider based on initiation of interaction by the occurrence of a Provider TP4 Reaction Primitive. The Translator to Provider direction, the Provider TP4 Petition Primitives, will be presented first. This will be followed by the Provider to Translator direction, based on initiation of activity by the Provider TP4 Reaction Primitives.

4.2.1 TP4 Provider Petition Primitives

Service Petition Primitives cross from the Translator to the Provider. There are five Petition Primitives that need to be considered.

4.2.1.1 T-CONNECT Request (CNRQ)

The calling TSU invokes this primitive to establish a TC. The TSU specifies the destination address, its desired QOS, and its desire for the use of expedited data service, along with the possibility of sending user-data.

The Translator will issue a CNRQ to the Provider upon receiving an ACO or an AOD from the Origin TSU. The Translator will never issue the expedited data service request to the Provider, but it will include user-data if an AOD was received from the TSU.

4.2.1.2 T-CONNECT Response (CNRS)

The called TSU invokes the CNRS Petition Primitive to tell the initiator that it is willing to establish a TC. The called TSU may optionally transfer a limited amount of user-data with this primitive or indicate the expedited data option or both. It includes its address and QOS agreement.

The Translator will issue a CNRS to the Provider when it receives a CNRQ from the Provider if the called TSU has previously issued an UPO, a SPO, an ACO, or an AOD where the specified address is the Target TSU.

4.2.1.3 T-DATA Request (DARQ)

The TSU uses this primitive to inform the Provider of a desire to transmit user-data.

The Translator will issue the DARQ to the Provider upon receipt of a SND from the Origin TSU.

4.2.1.4 T-EXPEDITED-DATA Request (EDRQ)

The TSU uses this primitive to indicate to the Provider that it wishes to send expedited data on a Transport connection. The connection must have been established with expedited data usage negotiated. Expedited data is queued ahead of previously sent messages.

Since TCP does not support expedited-data, the Provider will not be called upon to provide this service. Hence, during the negotiation of service, the Translator must not request expedited data usage. The Origin TSU must concur with the Translator's request to omit that service in the standard set of conforming procedures. Therefore, no changes are necessary at the TSU when this service is denied.

4.2.1.5 T-DISCONNECT Request (DSRQ)

The TSU calls this function either to disconnect an established Transport connection or to refuse an incoming connect indication. User-data may optionally be included in the primitive.

The Translator will issue the DSRQ to the Provider upon receipt of a CNRQ for a TSU that is not in the correct state, upon receiving an ABT from the TSU, or upon receiving a CLS from the TSU. In the case of the CLS, all outstanding messages must be acknowledged before the DSRQ is issued, but it will not accept more message traffic from the TSU for that named TC.

4.2.2 TP4 Provider Reaction Primitives

Five Reaction Primitives enable TP4 to inform the TSU of user connection status, data delivery, and connection termination. Once again it will be the Translator's responsibility to convert the TP4 Reaction Primitives from the Provider, and issue correct TCP Reaction primitives to the TSU or TP4 Petition primitives to the Provider depending on the situation.

4.2.2.1 T-CONNECT Indication (CNIN)

The Provider uses this primitive to communicate to the TSU that a connection is being requested. The Provider specifies the address, whether expedited traffic is exercisable, the QOS, and possibly some user data.

The Provider receives this primitive to indicate that an Origin TSU wants to establish a TC with a Target TSU. The Provider will respond to the translator with a CNRS or DSRQ depending on the state of the Target TSU.

4.2.2.2 T-CONNECT Confirm (CNCF)

The Provider uses this primitive to communicate to the TSU that a connection has been completed. The address of the called TSU, and the QOS agreed upon are included as parameters, and optionally the status of the expedited data option and TSU user-data.

When the Translator receives a CNCF from the Provider, it will issue an OSC to the Origin TSU.

4.2.2.3 T-DATA Indication (DAIN)

The Provider uses this primitive to notify the TSU that data has been received on the connection.

The Translator will issue a DLV to the TSU.

4.2.2.4 T-EXPEDITED-DATA Indication (EDIN)

TP4 uses this primitive to notify the TSU that expedited data has been received on the connection.

Since the TCP/IP does not support expedited traffic as explained in 4.2.1.4, the Translator will not ever receive a EDIN because the Origin TSU Translator will never accept an EDRQ.

4.2.2.5 T-DISCONNECT Indication (DSIN)

The Provider issues this primitive to communicate a disconnect indication to the TSU. The disconnect reason is included with the option of including user-data.

The Translator will issue an ABT to the TSU. If data is included the Translator must first issue a DLV, and then the ABT.

4.3 Translator Sequenced Interchange Model

The Translator uses the Petition and Reaction Primitives for the functions of establishing, maintaining and releasing transport connections. This is achieved for any TCP TSU over the TP4 based network. The following sequences of transport level service primitives are the ordered procedures required to achieve the previously listed activities for the TCP/IP networks found in Chapter 2. These sequence items are required in the TCP/IP model, therefore they will be the driving functions that will be translated and supported on the TP4 network using the TP4 primitives.

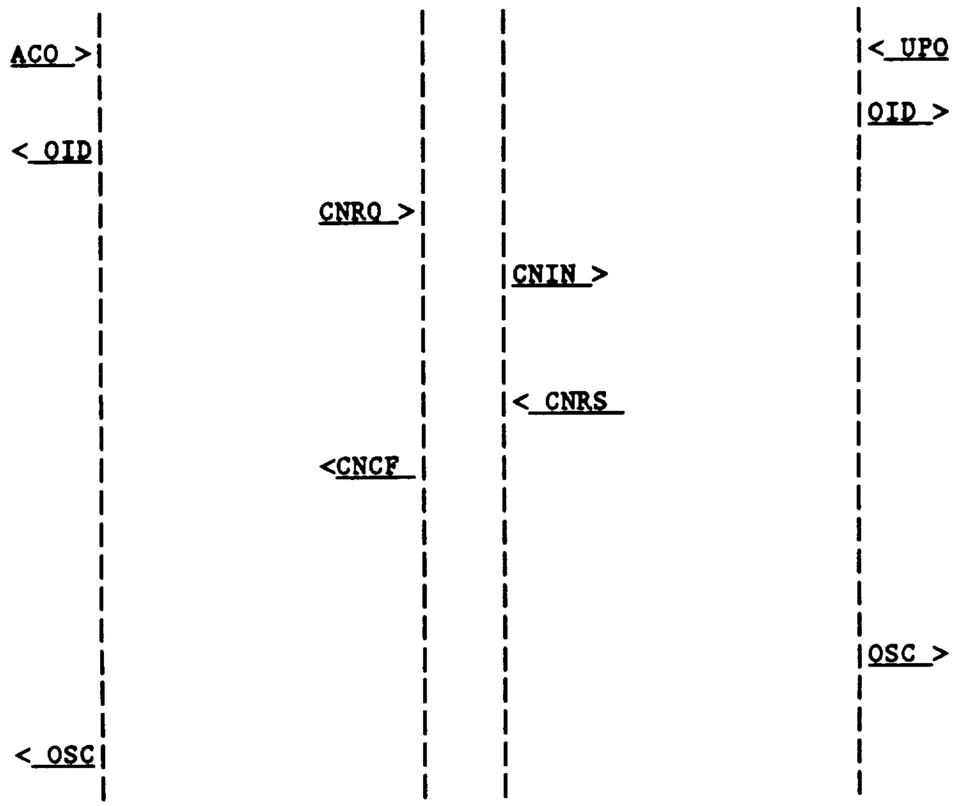

FIGURE 4-1. SUCCESSFUL TRANSPORT CONNECTION ESTABLISHMENT (PASS OPEN)

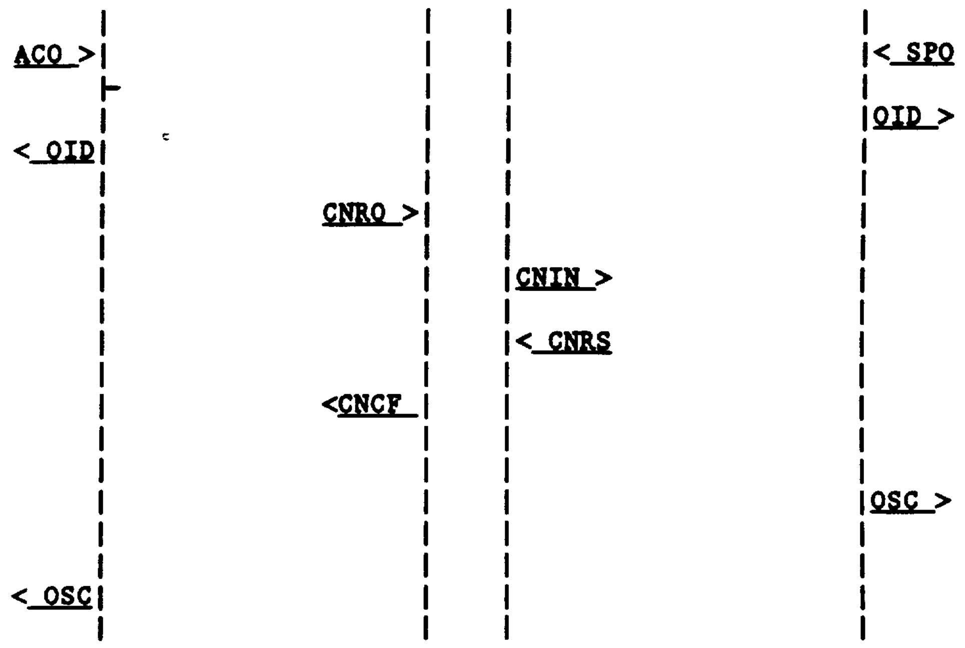

FIGURE 4-2. SUCCESSFUL TRANSPORT CONNECTION ESTABLISHMENT (SPEC OPEN)

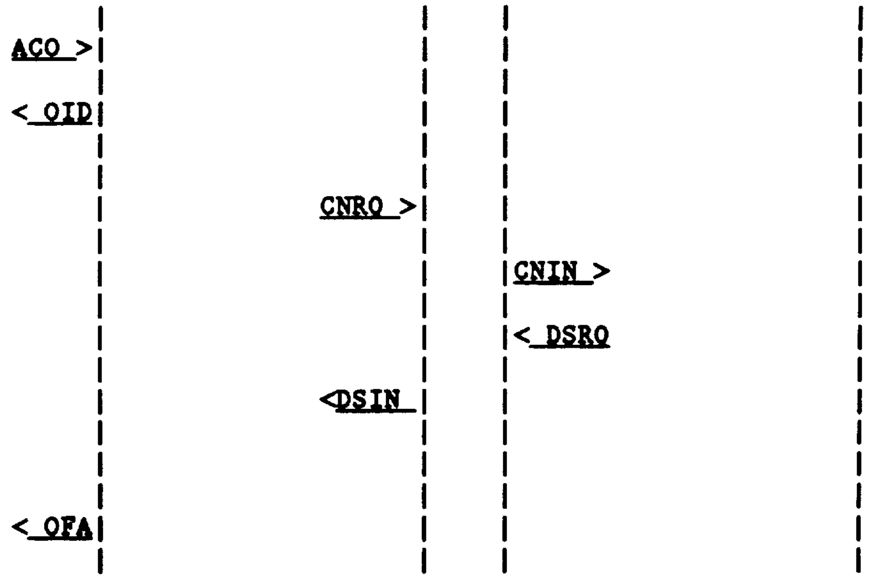

FIGURE 4-3. REJECTION OF TRANSPORT CONNECTION ESTABLISHMENT (NOT OPEN)

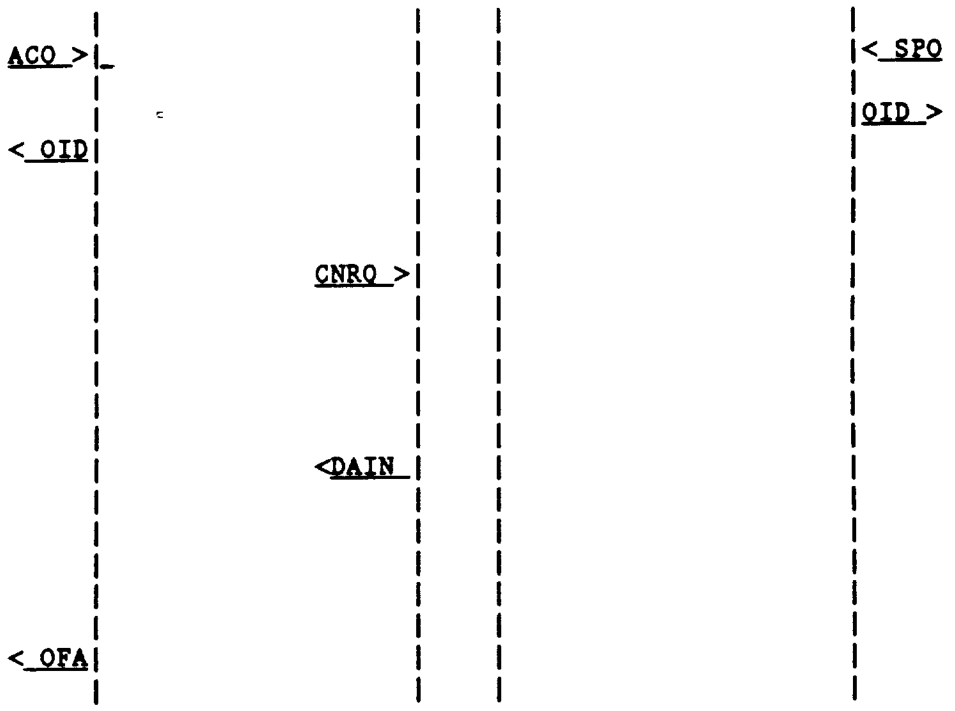

FIGURE 4-4. REJECTION OF TRANSPORT CONNECTION ESTABLISHMENT (PROVIDER)

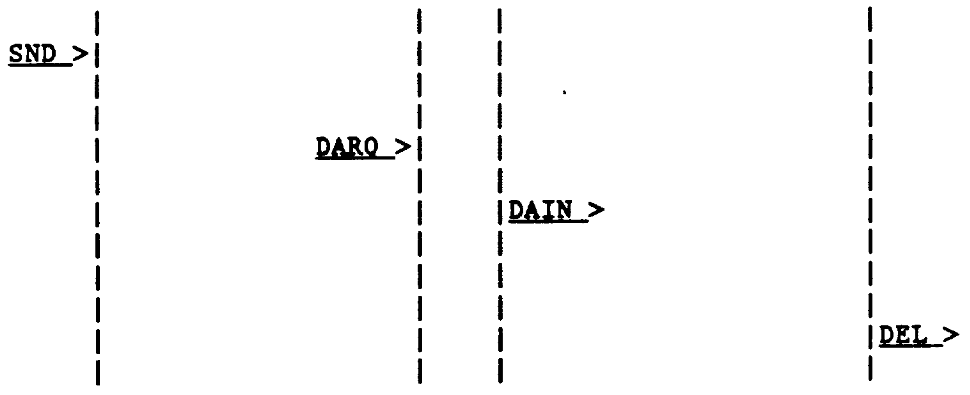

FIGURE 4-5. NORMAL DATA TRANSFER

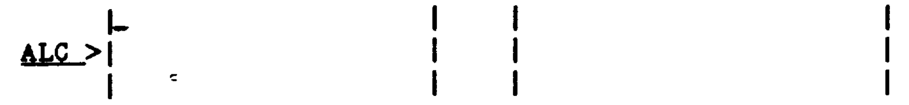

FIGURE 4-6. TRANSPORT SERVICE PETITION TO PROVIDER

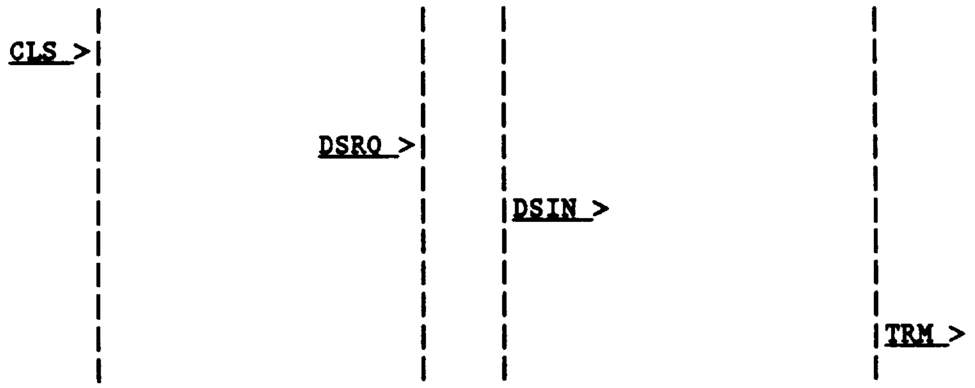

FIGURE 4-7. GRACEFUL CLOSE BY USER

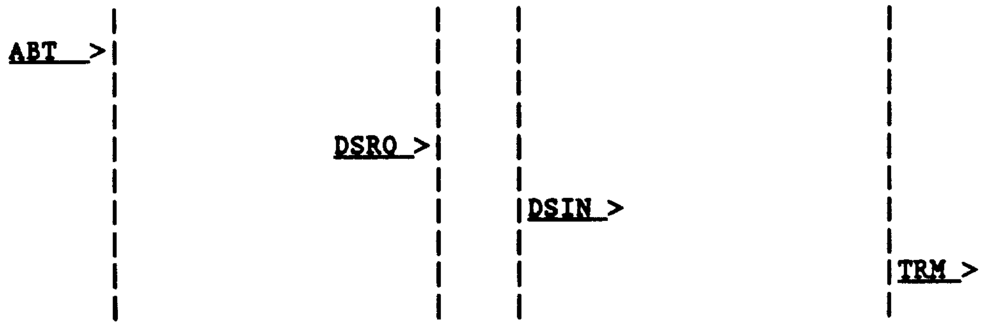

FIGURE 4-8. ABRUPT CLOSING BY USER

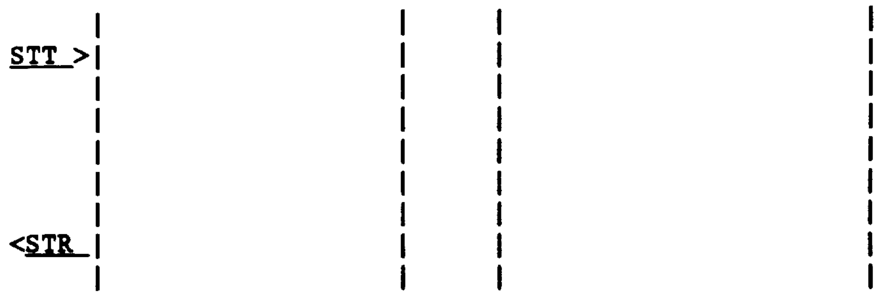

FIGURE 4-9. TRANSPORT SERVICE USER TO PROVIDER

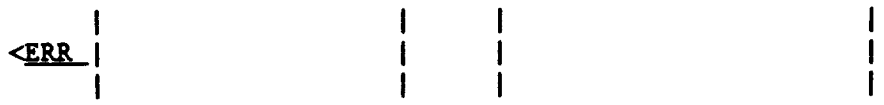

FIGURE 4-10. PROVIDER TO TRANSPORT SERVICE USER

5.0 PROOF-OF-CONCEPT DEMONSTRATION CODE

The initial phase of the research involved the theoretical investigation using standards, procedures, and generating abstract primitives for use in a Translator model. The goal was to determine whether an interface for TCP/IP application users could be provided that would allow TCP/IP based applications to operate across an OSI/CCITT based network. As was shown in prior sections of this report, a theoretical Translator model was produced which revealed that an actual coded TCP/TP4/TCP Translator could be generated. The following sections detail SwRI's efforts towards actually implementing the TCP/TP4/TCP Translator and the degree of success accomplished with this effort. This chapter is an outline of this work and should be understood that it was completed to prove out the concept on a real network.

The appendices contain the SwRI developed code and several of the test programs that were used in the development process. The following sections outline in fine detail the efforts that were required to complete the verification process. It is hoped that the general research information of the earlier chapters and the details of this specific implementation in this chapter will together provide the necessary guidance for a developer who is attempting to solve a similar problem. The sections will generally be presented in a chronological order as the tasks were actually accomplished.

5.1 Initial tasks

The first step toward implementing the Translator was obtaining the specific system software and hardware elements which were supporting the target network, the NASA/JSC GPLAN. This phase of the project was necessary so that the modeling and testing would be run on an off-line replica GPLAN network (one active link between two workstations) which would eliminate development and operational conflicts with the MCC throughout the process. This effort can be also be a source of problems, because as the translator is developed, the replica network must continue to roll with the revisions and activities on the actual network. Also, since the MCCU workstations are under configuration management, development at NASA would have caused additional slow down problems. The tradeoff in a mission critical environment like NASA's, however, makes a replica network experimentation procedure mandatory.

5.1.1 RTU Network Source Code

The source code for the network was needed to support the replica as well as to be the target of the modifications. The first problem seen once the source code was obtained was that the C language interface to the TCP/IP services of the operating system, called Real Time Unix (RTU), does not always directly correspond to the primitives of the TCP specification. In some cases, there was a one-to-one relationship between the standard's primitives and RTU services, but in many cases this was not true. The standard primitive was eventually accomplished through several RTU processing steps. The RTU TCP/IP network software source code provided valuable information, although often in a cryptic fashion, into the implementation of the TCP specification.

At the outset of this research, one possible option was to insert the Translator into the RTU kernel (inside the operating system). In RTU, and in Unix in general, the TCP/IP services provider software resides in what is referred to as the operating system's kernel. One of the possible options of implementing this research was to replace the TCP/IP services in the RTU kernel with the Translator. This option was eventually rejected for several important reasons. Section 5.2.2.1 provides a complete discussion of the advantages and disadvantages of an "in-kernel" Translator.

The RTU RTnet-TCP network software version 2.4 source code was being used as the basis for the TCP/IP network activities. It is owned by Concurrent Computer Corporation and the research was conducted through a nondisclosure agreement between Concurrent and SwRI. Because of this, only the SwRI source code is available for publication in the appendixes of this report. The SwRI software interfaces are documented well enough that individuals can understand the processes occurring without revealing the Concurrent code or knowledge to the workings of their code. However, straight forward SwRI software modifications to their code are not provided in the appendix because the changes may provide unnecessary insight into the proprietary code. These modifications are unique to this source code and is of little general application use. With some thought, they can be duplicated by those who properly hold the source code. The RTU operating system is based on the BSD Unix source code which is readily available from the University of California at Berkeley.

5.1.2 <u>GPLAN Software</u>

NASA/JSC currently uses an IBM implementation of commercially obtained OSI ISO based software for the MCCU GPLAN. Obtaining the IBM modification's and their unique implementation of the software was also required. This was accomplished through a nondisclosure agreement established between NASA/JSC and IBM. Therefore, once again, no specific details of the IBM or the commercial software source code, or SwRI modifications could be revealed in this report. As above however, the SwRI translator's concepts, the Translator code, and the test functions are clearly documented, hence the specific commercial code information is unnecessary to follow the conclusions of this report.

5.1.3 <u>GPLAN Hardware</u>

At this point in the Translator development, efforts were expended in establishing the hardware replica GPLAN network link. The interface hardware boards, again commercial proprietary, that support the NASA MCCU most current version of the GPLAN segment were obtained and installed.

5.1.4 Replica GPLAN Link

Upon obtaining the required software and hardware components, the next task was the combination of the components into a working replica GPLAN link. The replica GPLAN link was established between a NASA owned, but located at SwRI, Masscomp 6600 workstation and an SwRI owned Masscomp 6350. The replica GPLAN link was situated between the two physically separated workstations to assure the Translator would be tested using TP4 and all software and hardware below TP4 that makes up the network. This insured that the Translator actually operated properly across the "ISO/CCITT wire," and was not simply a contrived software situation.

5.1.4.1 LLC Driver

The fist stage required to establish the replica GPLAN link was the installation of the Lower Level Communication (LLC) driver on the two host computers. The LLC driver provides the RTU interface to the Ethernet boards. This process was completed with the help of FACC, to make modifications to the LLC driver so that it would work properly in the SwRI unique Masscomp 6350. This particular fix was unique due to the different equipment units that were being used, and is not related to the Translator design process.

5.1.4.2 WEX-less GPLAN

The second stage in the establishment of a replica GPLAN link was the building of the IBM GPLAN software to work in a "WEX-less" environment. Neither of the Masscomp hosts used in the development and testing of the Translator used the NASA WEX environment. The GPLAN software was successfully built and an operating WEX-less replica GPLAN link was established between a Masscomp 6600 and a Masscomp 6350 at the SwRI facility. The GPLAN segment was verified using the IBM GPLAN Workstation Network Test Facility.

5.2 TCP/TP4/TCP Translator Code

Once the GPLAN software was built and the GPLAN segment was operational, development of the Translator was able to progress. The development of the Translator consisted of three main tasks:

1) The development of an interface to the TP4 daemon

2) The development of the Translator daemon

3) The development of the C language interface

These tasks are discussed in detail in sections 5.2.1, 5.2.2, and 5.2.3.

5.2.1 Interface to TP4 Daemon

The first software task in the development of the Translator was the development of an interface to the TP4 daemon, the serving object for the

ISO/CCITT network. The Translator model specified that the Translator should access the ISO protocol stack at the TP4 level. The standard Session layer interface to the TP4 daemon was used as a guide for developing the Translator interface to TP4, but the Translator's function is different than that of the Session layer. The Session layer serves as an intermediary between the Presentation layer software which sits above the Session layer and Transport layer, which sits below it. The Translator however must act as an end user of TP4, so a new TP4 interface library was developed which contained only the portions of the Session layer which were required.

The library which provides the interface to the TP4 daemon for the Translator grew significantly during the development. As the Translator test programs increased in utility and complexity, the requirements of the interface to the TP4 daemon increased in its utility and complexity. This increased complexity can be seen in the modifications which were necessary to support multiple VCs for a single process.

5.2.1.1 Initial Pruning

Development of the Translator library required the building of the necessary functions, and deleting the unnecessary ones, associated with the Session layer responsibilities of the ISO model. Since only the TP4 daemon interface was needed, the following Session layer functionality was trimmed while building the new library:

1) Session layer state transition functions,
2) MAP 3.0 network management interface functions, and
3) Timer functions and spawn of "sesstime" program.

Only the portions of the Session library which provide the interface to the TP4 daemon were transferred in the new SwRI library functions. This library remains for the most part consistent with the methodology used in the Session library. The modifications that were necessary are detailed in following sections. The effort to maintain the methodology contained in the Session library was done to ensure compatibility with future versions of the IBM TP4 daemon and other GPLAN software as the future versions were rolled in. The scaled down TP4 daemon interface library is contained in libtp4_if.a. The C language source files and the Makefile for the Translator TP4 daemon interface are contained in the /Lan/Translator/TP4 subdirectory.

5.2.1.2 Multiple TSAPs in a Single Process

During the implementation of the RTU TCP/IP network programs it was determined that the interface to TP4 would need to be modified to support multiple Virtual Circuits (VCs) in a single program. Many of the RTU TCP/IP network programs use two simultaneously established sockets (based on the VCs). One socket is used to transfer commands and status information, and the other socket is used to transfer the raw data. The RTU network program remote login (rlogin) is an example of one of the programs that requires two VCs. However, and seemingly contrary to the standard, the Session layer interface to TP4 and the LAN daemon did not support the usage of two VCs between a single Session layer program and the TP4 daemon.

With some outside consultations with the original code writers, SwRI was able to make the modifications to the Translator TP4 daemon interface to support multiple VCs.

5.2.1.2.1 Message Queue IDs and Registration

The TP4 daemon receives requests and returns status to the Session layer via Unix message queues. The TP4 daemon maintains a message queue for each of the VCs, and each message queue is referenced by a unique identification number. In the IBM implementation, the message queue ID used was the Unix process ID. The scheme works for cases where only a single process uses a single VC. In the case where a process has multiple VCs, a different unique identifier must be established for each VC, and this was not supportable. In the TCP/IP protocol, the port number uniquely identifies the VCs, so it was determined that the VC port number should be used as the message queue identification number.

The use of the VC port number as the message queue identification number necessitated several changes to the TP4 daemon interface. The routines which establish connections to the TP4 daemon now had to determine a unique port number and to pass this port number to the routines which allocate and initialize the Unix message queue. Previously, the routines which allocated the message queues determined the process id via a Unix system call. In other words, the routine which called the message queue allocation routines did not need to pass in the message queue identification number. It was determined locally each time from the standard Unix information.

To support the use of the port number and the message queue identification number, the calls to the routines init_session() and init_squeue() were modified to contain the additional port number variable.

5.2.1.2.2 Event Queues

The TP4 daemon interface maintains a queue of the events that are returned from TP4 during processing. With the single VC scenario, this event queue could be declared globally within a single process. In a multiple VC scenario, multiple event queues were required to be instituted. When a connection is established from one TSU to another TSU, TP4 returns an event which contains a machine pointer (labeled machp). This pointer is determined by TP4 and is used by TP4 to identify each of the VCs that it is currently servicing. Subsequent calls to the TP4 daemon after connection establishment must contain this unique machp so that TP4 can determine for which VCs the request was made. Transport Service Users send events to the TP4 daemon via a single queue, so machp can be used by the TP4 daemon to determine for which of the VCs the event is targeted.

Multiple VCs could be maintained in a single event queue once a relationship is established between the TP4 machp and the corresponding VC. Unfortunately, during connection establishment the machp and VC relationship does not yet exist. If a process establishes two VCs at the same time, the process cannot determine which VC is established by TP4 on the basis of the information in the connection event structure. For this reason, multiple VCs within a single process require instituting multiple event queues.

The addition of multiple event queues to the TP4 interface library required that many of the calls to the interface routines also include an event queue pointer. These routines were modified to contain the proper event queue pointer.

5.2.1.2.3 TRANslator Daemon

The Session layer interface uses the process identification (pid) to identify its various resources, such as message queues. These resources are registered during their allocation with LANdaemon, which periodically ensures that resources which are active belong to processes which have not terminated are released back to the Operating System. One of LANdaemon's function's is to serve as the housekeeper for resources. LANdaemon reclaims resources (message queues, shared memory segments, semaphores, etc.) which are left active by terminated processes. One of the impacts of allowing multiple VCs, is that the TP4 interface routines can no longer use their pid to identify their resources. Therefore these resources could no longer be registered with LANdaemon. For this reason, and several others which are discussed in later sections, a Translator equivalent to LANdaemon was created.

The TRANslator daemon, or TRANd, was created to perform the resource housekeeping previously performed by LANdaemon. The resources used by the interface to TP4, especially message queues, are usually not in abundance and it is important that they be reclaimed as quickly as possible. The TRANd operates slightly different than the LANdaemon. The LANdaemon registers resources on a process basis. The TRANd registers resources on a File Descriptor basis due to the Translator's requirement to act like a TCP/IP type socket handler. In the Unix TCP/IP implementation, sockets are maintained within the kernel as standard file descriptors and these file descriptors can be shared between many processes. This necessitated that the TRANd allocate and reclaim resources on a file descriptor basis instead of the process basis which is used within LANdaemon.

5.2.1.2.4 Multiple VC Demonstration

The multiple VC capability of the Translator TP4 daemon interface has been demonstrated using two test programs: recv2 and send2. These two programs work in tandem to open two sockets and then alternately transfer data across the two sockets. The complete listings for the test programs recv2 and send2 can be found in the appendices.

5.2.2 C language library of TCP calls

Once an interface library to the TP4 daemon was organized and coded, development of the actual Translator library routines could begin. The Translator library has to contain a one-to-one functional replacement for each of the C language TCP/IP functions that is available to the user within the RTU operating system. The Translator is designed to transparently replace the TCP/IP network functions with new functions that use similar TP4 instructions, or combinations of TP4 instructions, out of the TP4 instruction set as the transport level protocol. Once this is accomplished, as was shown in the theoretical development, the RTU TCP/IP network programs, or any TCP/IP socket based program, could begin to operate across the ISO based GPLAN without modification.

5.2.2.1 In-kernel vs External Translator

It was originally proposed that the Translator would either replace or augment the TCP/IP services of the RTU kernel. The concept was that if the Translator were placed within the RTU kernel, then any TCP socket based program would operate across the GPLAN without any modifications. Investigation of the RTU TCP/IP network software source code confirmed that this approach was a feasible research approach. That is, since the RTU TCP/IP functions were already linked into existing applications, they only contained a subroutine call to enter the RTU kernel. With this approach, little work is required to be done outside the kernel. Figure 5-1 is a flow diagram that illustrates the interaction.

```
TCP/IP User application ---> TCP function -------------------->
                                                               |
                              (kernel entry                    |
                              point only)               RTU kernel
                                                               |
                                                               |
TCP/IP User application <--- TCP function <--------------------

                              (values returned
                              from kernel are
                              returned to
                              application unmodified)
```

FIGURE 5-1. TCP FUNCTION TO RTU KERNEL FLOW.

This condition makes the kernel replacement of the TCP/IP functions feasible. If the TCP/IP functions contained more than the kernel entry point, then the applications which used the new kernel would have to be relinked with the replacement routines. Being feasible does not mean that it is appropriate. The same rational that determined the internal translator was feasible, yields the conclusion that a translator external to the kernel, but providing the TCP/IP replacement of the kernel, is also feasible. The advantages and disadvantages of each research strategy had to be scrutinized before the final design could be established and the implemented code generated.

5.2.2.1.1 In-kernel Advantages

There are several advantages in using an in-kernel Translator. First, an in-kernel Translator allows the TCP/IP based programs to operate over the ISO network with no modifications by the users. The same program executables which are used across the current TCP/IP network could operate through the ISO TP4 kernel without recompiling or relinking. This is a step more than the no software changes assumption, this means no action of any kind is necessary by the TCP/IP user community to utilize the translator. The internal kernel also allows standard input/output "stdio" and standard error "stderr" reports to be redirected over a socket because the redirection occurs within the kernel, and the external Translator can not simulate this function.

Second, the in-kernel Translator would most probably offer higher performance than would an external Translator. The in-kernel Translator would operate as part of the RTU operating system. As part of RTU, the process could operate with the execution privileges of the operating system instead of the standard execution privileges assigned to user processes that the external Translator would have.

Third, a Translator residing internal to the kernel would also mean that the Translator data structures would reside in the kernel's protected memory, instead of in the more volatile application memory area.

5.2.2.1.2 In-kernel Disadvantages

The in-kernel Translator also has several very important disadvantages that needed to be considered. First, an in-kernel Translator would be usable with only the specific hardware and RTU operating system software release with which it was developed. For example, an in-kernel Translator developed for the Masscomp 6600 RTU release 4.1A could not be used on any hardware other than a Masscomp 6600 and it must use that specific release of RTU software. The Translator kernel would require modifications to be integrated into another workstation or another Masscomp 6600 running a different version of RTU. Further, when the next release of RTU following version 4.1A is desired, extensive modifications to support the Translator may have to be accomplished in the new version. This is in fact the case for the next RTU version 5.0 because the new kernel is fully symmetrical and contains support for Streams which was not true of RTU 4.1A. Whenever work in the communication area occurs, the Translator would be in danger of having untested and inconsistent interactions with the RTU code. As the operating system internals are changed, the Translator would require revision. Therefore, an external Translator is consequently more portable to other hardware platforms and other versions of the RTU operating system. In fact, the external Translator developed has been demonstrated with two different versions of RTU and with two different types of Masscomp workstation systems. Specifically, the Translator is currently operating on a VME based Masscomp 6350 running version RTU 4.1A and communicating with a MULTI-BUS based Masscomp 6600 running version RTU 4.0.

Second, the TP4 code is already an external application to the kernel. Therefore an exit from the kernel to the TP4 application is necessary after the internal translation occurs. This leads to the conclusion that an external Translator is the natural operation domain for the TP4 applications.

Third, an in-kernel Translator also imposes a less favorable development environment. To place the Translator in the kernel, the kernel object files must be linked with other object files which compose the programs which is the kernel. This process is considerably more time consuming than the development of an external kernel which appears as just another application program. The entire set of files that composes the "/unix" program that makes up the kernel would require recompiling and testing.

Fourth, in-kernel development would most probably render the development machine unusable to other developers while the Translator development is in progress. This would be due to the large number of times that the machine would be not be available during the rebooting of the new operating system and the testing process that would be required during each version. Also the other users could be impacted by inadvertent coding errors introduced during the Translator development.

For the reasons stated above, it was determined early on that it would be in NASA's best interest to develop an external-kernel Translator. The Translator was prototyped and demonstrated as a library of TCP/TP4/TCP routines which could be linked with any TCP/IP socket based C language program. The source code and object files of the TCP/IP program require no modification to use the Translator. The completed object files of the program need only to be relinked with the Translator.

5.2.3 Translator Library

The Translator library generated is contained in the Unix library file called "libiso.a". This library is linked with a TCP/IP socket program. The Translator library is composed of all the standard TCP/IP network functions. The source files for the Translator library routines are located in the "/Lan/Translator/translator" directory. A review of the source files will show that a replacement for all of the TCP/IP network functions exist with the same calling sequence as the RTU C language version of the network functions. Figure 5-2 given below establishes the relationship of the model to its functions and the actual subroutines used to complete the action.

The translation from TCP/IP to TP4 is performed within the Translator library routines. As seen in Figure 5-2, the Translator library routines perform the following functions:

1) Establish a connection to the TP4 daemon and establish a shared memory segment which is used to transfer data to/from the TP4 daemon.

2) Establish a TP4 connection to another Transport Service User.

3) Transfer data from one TSU to another.

4) Close a Transport connection.

Model	Description	TCP/IP	ISO Translator
Active Open	Actively establish a connection	socket()	LANmat() init_buffers()
		connect()	init_session() TSUadd() UCONreq()
Passive Open	Passively open a connection waiting for an active open	socket()	LANmat() init_buffers()
		bind()	build TSAP/NSAP
		listen()	init_session() TSUadd()
		accept()	connection wait check_TP4_ queue() UCONres()
Send	Send data	send()	UDATreq()
Receive	Receive data	recv()	UDATrcv()
Close	Close a connection	shutdown()	TP4_msg(TDISREQ)
Abort	Abort	exit()	TP4_msg (DEACTIVATE)
Status	Get connection status	getsockname()	Provided locally
		getsockopt()	Provided locally

FIGURE 5-2. MODEL AND CODE RELATIONSHIPS

The Translator library also contains a replacement for several routines which are not normally associated with the TCP/IP network functions. These functions are required as a result of the way in which TCP/IP connections are managed within Unix and RTU. TCP/IP socket connections are maintained via the standard file descriptor mechanism used in Unix. File descriptors are typically used to read and write files, and to read and write standard input/output (stdio) and standard error (stderr). Unix and RTU maintain and reference TCP/IP socket connections via the file descriptor. As a result, all the functions which are typically associated with managing files by means of the file descriptor, must also be implemented in the Translator library. A survey of the Translator library will show that the C language functions such as: read(), write(), dup(), exit(), and fcntl() are implemented. These functions are not typically associated with TCP/IP functions. For example, the following functions are implemented in the Translator library, primarily due to the management of sockets via the file descriptor:

> The Translator contains a replacement for the exit() function. This function is used to terminate a program, and is not normally associated with TCP/IP sockets. One of the functions of exit() is to close and flush all file descriptors. As a result, a replacement for the exit() function was written which closes all of a program's TP4 connections.
>
> The Translator contains a replacement for the read() and write() functions. These functions are typically thought of as the functions which are used to read and write disk files. The recv() and send() functions are the functions which are typically thought of as the functions which are used to read and write sockets. Surprisingly, because sockets are maintained as file descriptors, the read() and write() functions can be used on sockets also.
>
> The Translator contains a replacement for the fork() function. The fork() function is used to create another executing copy of a program while the program is executing. A feature of the fork() function is the availability of the parent's file descriptors to the child/spawned process. Due to the fact that sockets are maintained as file descriptors, the child process created by a fork() can also utilize the parent's TCP/IP sockets. As a result, a replacement for the fork() function was also needed so the child process created by a fork() would have access to the TP4 connections established by its parent.

5.2.4 Virtual Circuit (VC) Addressing

Both the TCP/IP and ISO transport level protocols contain a mechanism for identifying/addressing a VC. In the TCP/IP protocol, a transport level address consists of an Internet number and a port number. Figure 5-3 shows how a VC is addressed using both the local and remote transport addresses.

TCP/IP VC Identifier

Local TCP/IP Address
1. Local Port
2. Local Internet Number

Remote TCP/IP Address
1. Remote Port
2. Remote Internet Number

FIGURE 5-3. TCP/IP VC IDENTIFIER

In the CCITT/ISO protocol, a transport level address is referred to by a TSAP. The TSAP is a user defined id which is passed from the transport service user to TP4. The Translator model requires that the ISO TSAPs contain the same format as the TCP/IP address. This addressing scheme works perfectly as long as the TSAP can represent any valid TCP/IP address. And in fact, due to the number of bytes which are available to construct a TSAP, any valid TCP/IP VC identifier can be represented. As a result, the Translator identifies VCs by the standard TCP/IP address pairs.

5.2.5 Host Addressing

The Translator library must construct both a valid TSAP and NSAP for the target TSU that it wishes to connect to. Both the TSAP and NSAP addresses contain the ethernet number of the local and remote ethernet boards. The Translator uses a local host table to convert Internet addresses to ethernet addresses. This host table is located in the file called "/Lan/Config/hosts" and has the format "internet number ethernet number host name."

A host table file must exist on every host which uses the Translator and this table must contain a valid entry for every host which is accessed via the Translator. This table is used by the TCP/IP replacement routines gethostmame() and gethostbyaddr().

5.2.6 TRANd daemon

The TRANslator daemon TRANd was developed as a result of several factors. Primarily, the decision to not replace the TCP/IP services of the RTU kernel with the Translator necessitated that a host level controller be implemented. However, the modification of TP4 daemon interface to support multiple VCs also necessitated a host level controller similar to LANdaemon.

5.2.6.1 Replacement for LANdaemon

The Translator library routines support the simultaneous use of two VCs within a single process. To implement this functionality, the calls to register a Session level process's resources with LANdaemon were removed.

The LANdaemon is designed to use the process id to locate the message queues and other resources of the processes that have terminated. The Translator routines use port numbers to create and reference message queues, so a replacement for LANdaemon was needed. The Translator daemon process performs some of the same functions as LANdaemon but only for Translator linked processes and resources. The TRANd maintains a table of processes similar to LANdaemon. If TRANd determines that a process has terminated without removing its message queues, the TRANd removes the message queue using the TCP/IP port numbers of each VC.

5.2.6.2 Multiple Translator Processes

The TRANd also allows multiple processes which use the Translator routines to coexist in a single host. The TRANd allocates and initializes a shared memory segment which is accessed by every Translator linked routine. The shared memory segment maintains the state of each Translator VC for the entire host. This allows port numbers to be assigned in the Translator routines in the same manner as they are assigned in the TCP/IP routines. The shared memory segment created by TRANd also allows multiple processes to access the same set of file descriptors (sockets). This becomes necessary during the fork() and exec() functions which may allow separate processes to utilize the same file descriptor (socket).

5.3 Prototype Results

The Translator prototype has demonstrated that a TCP/TP4/TCP translator can be implemented. The Translator has been successfully demonstrated with several test programs, including the Unix TCP/IP network program tftp and its server daemon tftpserver. The Translator has been demonstrated within a single host and across a thick segment ethernet GPLAN between two hosts.

The first test programs which were used to test the Translator and TP4 interface library were: recvtcp.c and sendtcp.c. These programs were taken from the RTU Programming Manual and are a perfect example of how the C language interface to TCP should be setup and operated. These programs have been linked with the Translator and demonstrate that a TCP/IP based program can be operated over an ISO based network with no modifications.

The second set of test programs which were implemented were the RTU programs: remote time (rtime) and its server daemon "timeserver." This program was the first of the Unix TCP/IP network programs to be implemented and tested.

The third set of test programs which were demonstrated with the Translator were the RTU programs: Trivial File Transport Protocol (TFTP) (routine "tftp") and its server daemon "tftpserver."

As discussed previously in this report, the external Translator does inherently contain limitations over an in-kernel Translator. The external Translator requires that every program be linked with the Translator library if it is to operate over the CCITT/ISO based network instead of the TCP/IP based network.

6.0 CONCLUSION

6.1 Theoretical Development

A level of knowledge has been stimulated by this project that has public purpose and has extended the basic knowledge and understanding in technical communications methods used in space operations. The TCP/IP protocol is a widely used computer-communication procedure, and its transport level, TCP, supports connectivity throughout the nation's computer-communication networks. Internationally, however, the OSI stack has seen increased implementation, and hence ISO/CCITT based standards are becoming the reasonable substitute of choice for TCP/IP applications. Until systems can fully incorporate the new and developing ISO/CCITT standards, this TCP/TP4/TCP prototyped Translator process demonstrated a method to bring about restructuring of systems supporting the OSI transport layer, while allowing the upper-level protocols and applications to remain static.

Chapters One through Three provided the basis for the developed abstract Translator primitives created in Chapter Four. All four of these chapters are considered within the theoretical solution because they present the communication standard's abstract primitives that were applied as the cornerstone for the development of the Translator's set of abstract primitives. Within this framework, it was clearly demonstrated that a set of procedures could be developed that meets the objective of using TCP application programs to communicate over a TP4 based network.

There were a few TCP functions, such as the URGENT marking of a message, that could not be accomplished within the standard ISO/CCITT network because of the lack of a similar mechanism in the TP4 protocol. It was recommended that Translator-to-Translator maneuvers could be developed to support these kinds of activities, however this appeared beyond the scope of the current activity. Instead, the theoretical Translator responded to these kinds of requests by either using a legal TCP denial for the TCP/IP service request or to generate a local value to simulate the response to an end-to-end request. This allowed the TCP/IP application to work within the constraints of the TP4 network.

The primary assumptions of providing the movement to ISO/CCITT based systems at a later date and no software changes required by the users were not violated by the above system, because all code implementations were designed to occur within the Translator. Therefore the user should be unaware of the Translator's existence, which is necessary to meet both assumptions.

6.2 Proof-of-Concept

The research work reported in Chapter Five demonstrated both the capacity to implement the theoretical Translator and some of the difficulties in taking on such a challenge. The accomplishment of the transfer of actual working files over the TP4 based network using application programs written strictly for the TCP interface clearly displays that the objectives can be met in real world networks. The general message passing programs that were available to demonstrate a standard C language interface to TCP operated properly when setup with the Translator. The utility remote time ("rtime") successfully demonstrated a typical standard service that is available to TCP/IP users. Finally, file transfers using the more complex TCP/IP application TFTP was achieved on the TP4 based network.

The primary assumption to allow for future movement to the exclusive use of the CCITT/ISO standards once the application programs no longer need the translator is inherent in the noninterfering ISO/CCITT interaction established by the SwRI Translator code and library routines. Once the TCP/IP systems are no longer accessing the network, the Translator could be withdrawn in a totally transparent manner. The movement of these applications to the ISO/CCITT network standards is not impeded in any way by the Translator's existence.

The other primary assumption that the application programs would require no software alterations was also absolutely met. The interface is truly a seamless crossover from the user's point of view. The requirement imposed on the user to simply relink their TCP based object code to the SwRI developed Translator library is as transparent an action as could be conceived to implement a system that holds this level of complexity.

6.3 Further Research Considerations

There were many additional unexpected considerations that arose during the implementation of the TCP/TP4/TCP Translator. The objective of developing the SwRI demonstration code was to establish the feasibility of performing the Translator function within an actual system. Two important issues determined to be candidates for further study are the consequences of nonconformity to the standards and the Translator's impact on network performance.

6.3.1 Nonconformity to Standards

As was seen, when the theoretical translator was employed in a system that has nonconformities to the standards, it demanded out-of-model responses during the implementation. The impact of nonconformity will always be more prominent when strictly commercial off-the-shelf (COTS) implementations are considered the basis. The designers of these systems expect to be able to integrate the standardized systems without problems, right off the shelf. This is more than an expectation, it is usually an essential need, because these designers normally have little to no detailed knowledge of the source code that supports their system. They are expecting the COTS products to interwork as advertised. Obviously, options and various interpretations of a standard complicate this issue. This factor caused us to reach another conclusion about this research. If a heterogeneous system of manufacturers implement a set of standard protocols used for the network communication standards, it can be expected to eventually fail because of the different manufacturer interpretations and optional development inconsistencies. This is especially apparent during follow-on "in-standard" expansion efforts of the network even when all of the manufacturers claimed conformance to the same protocols. Complex communication system networks will always benefit when they are throughly tested against conformity to the standards.

6.3.2 Performance Impact Issues

In this case, the proof-of-concept activities were applied to a network that operates in a bursty message mode, with peaks in the demand on the network. During the proof-of-concept evaluation, there was no indication of slowdown or bogging of the network because of the existence of the Translator external to the operating system. The CCITT/ISO networks are reputed to be faster than the TCP/IP networks, so it is reasonable to expect that little slowdown would be apparent when comparing "a TCP/IP only network" against the "TCP/TP4/TCP translator network". However, analytical measurements to compare performance was not accomplished in this study. Before a final installation of a translator system is completed, it would be a valuable exercise to spend research time in the area of expected network performance alteration due to addition of such a translator. Not only would the translator performance impact result, but a basic standard of performance of the network could be characterized for future load study comparisons.

APPENDIX A: ACRONYMS

APPENDIX A: ACRONYMS

ABT	Abort
ALC	Allocate
AOD	Active Open with Data
ACO	Active Open
ANSI	American National Standards Institute
CCITT	International Telegraph and Telephone Consultative Committee
CLG	Closing
CLS	Close
CNCF	T-Connect Confirm
CNIN	T-Connect Indication
CNRQ	T-Connect Request
CNRS	T-Connect Response
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CHC	Channel capacity
DAIN	T-Data Indication
DARQ	T-Data Request
DLV	Deliver
DOD	Department of Defense
DSIN	T-Disconnect Indication
DSRQ	T-Disconnect Request
EDIN	T-Expedited Data Indication
EDRQ	T-Expedited-Data Request
ERR	Error
GPLAN	General Purpose LAN
IEEE	Institute of Electrical and Electronic Engineers
I/O	Input/Output
ISO	International Organization for Standardization
ITU	International Telecommunications Union
LAN	Local Area Network
LAN	Local Area Network
JSC	Johnson Space Center
MAN	Metropolitan area network
MCC	Mission Control Center
MCCU	MCC Upgrade
NASA	National Aeronautics and Space Administration
NIST	National Institute for Standards and Technology
NRC	National Research Council
NSAP	Network Service Access Point
OFA	Open Failure
OID	Open ID
OSC	Open Success
OSI	Open System Interconnection
QOS	Quality of Service
RTU	Real Time Unix
SAP	Service Access Points
SND	Send
SPO	Specified Passive Open
STR	Status Response

STT	Status
TC	Transport Connection
TCP/IP	Transmission Control Protocol/Internet Protocol
TFTP	Trivial File Transport Protocol
TP	Transport Protocol
TS	Transport Service
TSAP	Transport Service Access Point
TSDU	Transport Service Data Unit
TRM	Terminate
ULP	Upper Layer Protocols
UPO	Unspecified Passive Open
VA	Virtual Address
VC	Virtual Circuit
VS	Virtual Storage
WAN	Wide Area Network

APPENDIX B: REFERENCES

APPENDIX B: REFERENCES

Black, Uyless, "Data Networks, Concepts, Theory, and Practice," Prentice Hall, New Jersey, 1989.

Concurrent Computer Corporation, "RTU Programming Manual," Revision C, 1988.

Comer, Douglas, "Internetworking with TCP/IP," Prentice Hall, New Jersey, 1988.

Faulk, S.R., D.L. Parnas, "On Synchronization in Hard-Real-Time Systems," Communications of the ACM, Vol. 31, No. 3, March 1988, pp. 274-287.

IBM Corp., IBM 3081 Functional Characteristics, IBM Pub. No. Ga22-7076-7, Seventh Edition, 1986.

Kearney, M. W., "The Evolution of the Mission Control Center," Proceedings of the IEEE, Vol 75, No. 3, March 1987.

Kusmanoff, A. L., "Real Time Bearing Estimation in a Multi-source Environment Using Multi-processor, Multi-algorithmic Acceleration," Ph.D. Dissertation, Oklahoma State University, May 1989.

Perry, T.S., Zorpette, G., "Supercomputer Experts Predict Expansive Growth," IEEE Spectrum, February 1989, pp. 26-33.

Schneidewind, N.F., "Distributed System Software Design Paradigm With Application to Computer Networks," IEEE Transactions on Software Engineering," Vol. 14, No. 4, April 1989, pp. 402-412.

Shatz, S.M., J. Wang, Tutorial: Distributed-Software Engineering. Washington, D.C.: IEEE Computer Society Press, 1989, pp. 58-59.

Stallings, William, "Handbook of Computer-Communications Standards, Volume 1, The Open Systems Interconnection (OSI) Model and OSI-Related Standards, Howard Sams & Company, 1987.

Stallings, William, "Handbook of Computer-Communications Standards, Volume 3, Department of Defense (DOD) Protocol Standards, Howard Sams & Company, 1987.

APPENDIX C: TRANSLATOR TEST ROUTINES

```
#
#  Generate the TCP/ISO/TCP translator test routines.
#
#   TJB - Southwest Research Institute
#         San Antonio, Texas
#

ALL     =   sendtcp recvtcp recv2 send2

CFLAGS  =   -c
#CFLAGS =   -c -g -DDEBUG

LDFLAGS =   -g

LIB     =   /Lan/Libs/libiso.a
LIBDIR  =   /Lan/Libs
LIBS    =   $(LIBDIR)/libiso.a \
            $(LIBDIR)/libtp4_if.a \
            $(LIBDIR)/libsystem.a \
            $(LIBDIR)/liblanbuffer.a \
            $(LIBDIR)/liblanutil.a

INCDIR  =   -I/Lan/Src/Lan/Include \
            -I/Lan/Src/Lan/Include/sesstests
INCL    =   $(INCDIR)

all         : $(ALL)

#
#  Translator test programs - sendtcp
#

sendtcp.o:  sendtcp.c $(LIB)
            $(CC) $(CFLAGS)  -o $@ sendtcp.c

sendtcp:    sendtcp.o $(LIB)
            $(CC) $(LDFLAGS) -o $@ sendtcp.o $(LIBS)
#           $(CC) $(LDFLAGS) -o $@ sendtcp.o

send2.o:    send2.c $(LIB)
            $(CC) $(CFLAGS)  -o $@ send2.c

send2:      send2.o $(LIB)
            $(CC) $(LDFLAGS) -o $@ send2.o $(LIBS)

#
#  recvtcp
#

recvtcp.o:  recvtcp.c $(LIB)
            $(CC) $(CFLAGS)  -o $@ recvtcp.c

recvtcp:    recvtcp.o $(LIB)
            $(CC) $(LDFLAGS) -o $@ recvtcp.o $(LIBS)
#           $(CC) $(LDFLAGS) -o $@ recvtcp.o

recv2.o:    recv2.c $(LIB)
            $(CC) $(CFLAGS)  -o $@ recv2.c

recv2:      recv2.o $(LIB)
            $(CC) $(LDFLAGS) -o $@ recv2.o $(LIBS)
```

```
/*
 *      send2 - program to send data to two sockets.
 *
 *      This program is an adaptation of recvtcp.c which was taken
 *      from the RTU Programming Manual.  This program is used as
 *      an example of two TCP/IP socket connections across an ISO
 *      LAN simultaneously.
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <netdb.h>

#define ERR     1
#define NO_ERR  0

struct sockaddr_in sinhim1 = { AF_INET };
struct sockaddr_in sinhim2 = { AF_INET };

struct hostent *hp1;
struct hostent *hp2;

int         fd1,
            fd2,
            count;
extern int  errno;
short       buf[512];

/*
 *  Main routine.
 */

main( argc, argv )

    int     argc;
    char    **argv;

{
    register int    i;

    /*
     *  Check command line argument count.
     */

    if ( argc != 5 )
        {
        fprintf( stderr, "\nUsage: sendtcp host port port count\n\n" );
        exit( ERR );
        }

    /*
     *  Make sure we can resolve host name.
     */

    hp1 = gethostbyname( argv[1] );
    if (! hp1 )
        {
        fprintf( stderr, "\nHost: '%s' not found\n\n", argv[1] );
        exit( ERR );
        }
    else
    hp2 = gethostbyname( argv[1] );

/*
 *  Build address structure.
 */

bcopy( hp1->h_addr, &sinhim1.sin_addr, sizeof(sinhim1.sin_addr) );
bcopy( hp2->h_addr, &sinhim2.sin_addr, sizeof(sinhim2.sin_addr) );

sinhim1.sin_port = atoi( argv[2] );
sinhim1.sin_port = htons( sinhim1.sin_port );

sinhim2.sin_port = atoi( argv[3] );
sinhim2.sin_port = htons( sinhim2.sin_port );

/*
 *  This is the number of messages to send.  Build a dummy message.
 */

count = atoi( argv[4] );
strcpy( (char *) buf, "Start of Message:     01234567890 :End of Message" );

/*
 *  Create both sockets.
 */

if ((fd1 = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
    perror( "\nsendtcp socket()" );
    exit( ERR );
    }

if ((fd2 = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
    perror( "\nsendtcp second socket()" );
    exit( ERR );
    }

/*
 *  Establish connection to remote host.
 */

if (connect(fd1, &sinhim1, sizeof(sinhim1)) < 0)
    {
    perror( "\nsendtcp connect()" );
    exit( ERR );
    }

/*
 *  Send messages to remote host using "write()".
 */

errno = 0;
for ( i=0; i<4; i++ )
    if ( write(fd1,(char *) buf,strlen(buf)) != strlen(buf) )
        break;

/*
 *  Check for error writing to remote host.
 */

if ( errno )
    {
    perror( "\nsendtcp: I/O error" );
```

```
            exit( ERR );
            }
        else
            printf("Sent %d records to remote host\n", count);

        /*
         *  Make second connection.
         */

        if (connect(fd2, &sinhim2, sizeof(sinhim2)) < 0)
            {
            perror( "\nsendtcp second connect()" );
            exit( ERR );
            }

        /*
         *  Write data to second socket.
         */

        errno = 0;
        for ( i=0; i<4; i++ )
            if ( write(fd2,(char *) buf,strlen(buf)) != strlen(buf) )
                break;

        /*
         *  Write data to first socket again.
         */

        errno = 0;
        for ( i=0; i<count; i++ )
            if ( write(fd1,(char *) buf,strlen(buf)) != strlen(buf) )
                break;

        exit( NO_ERR );
    }
```

```
/*
 *      recv2 - program to receive data from two sockets.
 *
 *      This program is an adaptation of recvtcp.c which was taken
 *      from the RTU Programming Manual.  This program is used as
 *      an example of two TCP/IP socket connections across an ISO
 *      LAN simultaneously.
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <netdb.h>

#define ERR     -1
#define NO_ERR  0

struct sockaddr_in sinhim1 = { AF_INET };
struct sockaddr_in sinhim2 = { AF_INET };
struct sockaddr_in sinme1  = { AF_INET };
struct sockaddr_in sinme2  = { AF_INET };

struct hostent *hp;

int          protofd1,
             protofd2,
             fd1,
             fd2,
             count, sinlen;
extern int   errno;
short        buf[512];
char         *hostname;

main()
{
    /*
     *  Create both sockets.
     */

    if ((protofd1 = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        {
        perror( "recvtcp: socket()" );
        exit( ERR );
        }

    if ((protofd2 = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        {
        perror( "recvtcp: second socket()" );
        exit( ERR );
        }

    /*
     *  Bind both sockets.
     */

    if ( bind(protofd1, &sinme1, sizeof(sinme1)) < 0 )
        {
        perror( "recvtcp: bind" );
        exit( ERR );
        }

    if ( bind(protofd2, &sinme2, sizeof(sinme2)) < 0 )
	{
	perror( "recvtcp: second bind" );
	exit( ERR );
	}

    /*
     *  Get socket 1 name.
     */

    sinlen = sizeof( sinme1 );
    if ( T_getsockname(protofd1, &sinme1, &sinlen) < 0 )
	{
	perror( "recvtcp: getsockname" );
	exit( ERR );
	}

    printf( "recvtcp bound to port %d\n", ntohs(sinme1.sin_port) );

    /*
     *  Get socket 2 name.
     */

    sinlen = sizeof( sinme2 );
    if ( T_getsockname(protofd2, &sinme2, &sinlen) < 0 )
	{
	perror( "recvtcp: second getsockname" );
	exit( ERR );
	}

    printf( "recvtcp bound to port %d\n", ntohs(sinme2.sin_port) );

    /*
     *  Listen on socket 1.
     */

    sinlen = sizeof( sinhim1 );
    if ( listen(protofd1, 0) < 0 )
	{
	perror( "recvtcp: listen" );
	exit( ERR );
	}

    /*
     *  Accept on socket 1.
     */

    fd1 = accept( protofd1, &sinhim1, &sinlen );
    if ( fd1 < 0 )
	{
	perror( "recvtcp: accept" );
	exit( ERR );
	}

    /*
     *  Read the data from the first socket.
     */

    errno = 0;
    for ( count=0; count < 4; count++ )
	{
	if ( read(fd1, (char *) buf, sizeof(buf)) <= 0 )
	    break;
	else
	    printf("DATA1: %s\n", buf );
    }

/*
 *  Listen on socket 2.
 */

sinlen = sizeof( sinhim2 );
if ( listen(protofd2, 0) < 0 )
    {
    perror( "recvtcp: second listen" );
    exit( ERR );
    }

/*
 *  Accept on second socket.
 */

fd2 = accept( protofd2, &sinhim2, &sinlen );
if ( fd2 < 0 )
    {
    perror( "recvtcp: accept" );
    exit( ERR );
    }

/*
 *  Read the data from the second socket.
 */

errno = 0;
for ( count=0; count < 4; count++ )
    {
    if ( read(fd2, (char *) buf, sizeof(buf)) <= 0 )
        break;
    else
        printf("DATA2: %s\n", buf );
    }

/*
 *  Read data from the first socket again.
 */

errno = 0;
for ( count=0; 1; ++count )
    {
    if ( read(fd1, (char *) buf, sizeof(buf)) <= 0 )
        break;
    else
        printf("DATA1: %s\n", buf );
    }

/*
 *  Print any error message.
 */

if ( errno )
    {
    perror( "recvtcp: I/O error" );
    exit( ERR );
    }

hp = gethostbyaddr( &sinhim2.sin_addr, sizeof(sinhim2.sin_addr),
                    sinhim2.sin_family );
if ( hp )
    hostname = hp->h_name;
    else
        hostname = "UNKNOWN HOST";

    printf( "recvtcp read %d buffers from %s port %d\n", count, hostname,
            ntohs(sinhim2.sin_port) );

    exit( NO_ERR );
}
```

```
/*
 *  sendtcp - program to send data to a socket.
 *
 *  This program is taken from the RTU Programming Manual
 *  and is used as an example of TCP/IP socket communication
 *  across an ISO LAN.
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <netdb.h>

struct sockaddr_in sinhim = { AF_INET };

struct hostent *hp;

int         fd, count;
extern int  errno;
short       buf[512];

main( argc, argv )

    int     argc;
    char    **argv;

{
    register int    i;
    register struct hostent *hp;

    if ( argc != 4 ) {
        fprintf( stderr, "\nUsage: sendtcp host port count\n\n" );
        exit(1);
    }

    hp = gethostbyname( argv[1] );
    if (! hp ) {
        printf("\nHost: '%s' not found\n\n", argv[1] );
        exit(1);
    }

    bcopy( hp->h_addr, &sinhim.sin_addr, sizeof(sinhim.sin_addr) );
    sinhim.sin_port = atoi( argv[2] );
    sinhim.sin_port = htons( sinhim.sin_port );

    count = atoi( argv[3] );
    strcpy( (char *) buf, "Start of Message:     01234567890 :End of Message" );

    if ((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror( "\nsendtcp socket()" );
        exit(1);
    }

    if (connect(fd, &sinhim, sizeof(sinhim)) < 0) {
        perror( "\nsendtcp connect()" );
        exit(1);
    }

    errno = 0;
    for ( i=0; i<count; i++ )
/*      if ( send(fd,(char *) buf,strlen(buf),0) != strlen(buf) ) */
        if ( write(fd,(char *) buf,strlen(buf)) != strlen(buf) )
```

```
                break;

        if ( errno ) {
            perror( "\nsendtcp: I/O error" );
            exit(1);
        }
        else
            printf("Sent %d records to remote host\n", count);

        exit(0);
    }
```

```c
/*
 *  recvtcp - program to receive data from socket.
 *
 *  This program is taken from the RTU Programming Manual
 *  and is used as an example of TCP/IP socket communication
 *  across an ISO LAN.
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <netdb.h>

struct sockaddr_in sinhim = { AF_INET };
struct sockaddr_in sinme  = { AF_INET };

int             protofd, fd, count, sinlen;
extern int      errno;
short           buf[512];
char            *hostname;
struct hostent  *hp;

main()
{
    if ((protofd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror( "recvtcp: socket" );
        exit(1);
    }

    if ( bind(protofd, &sinme, sizeof(sinme)) < 0 ) {
        perror( "recvtcp: bind" );
        exit(1);
    }

    sinlen = sizeof( sinme );
    if ( T_getsockname(protofd, &sinme, &sinlen) < 0 ) {
        perror( "recvtcp: getsockname" );
        exit(1);
    }

    printf( "recvtcp bound to port %d\n", ntohs(sinme.sin_port) );

    sinlen = sizeof( sinhim );
    if ( listen(protofd, 0) < 0 ) {
        perror( "recvtcp: listen" );
        exit(1);
    }

    fd = accept( protofd, &sinhim, &sinlen );
    if ( fd < 0 ) {
        perror( "recvtcp: accept" );
        exit(1);
    }

    errno = 0;
    for ( count=0; 1; ++count ) {
/*      if ( recv(fd, (char *) buf, sizeof(buf), 0) <= 0 ) */
        if ( read(fd, (char *) buf, sizeof(buf)) <= 0 )
            break;
        else
            printf("DATA: %s\n", buf );
    }
    if ( errno ) {
        perror( "recvtcp: I/O error" );
        exit(1);
    }

    hp = gethostbyaddr( &sinhim.sin_addr, sizeof(sinhim.sin_addr),
                        sinhim.sin_family );
    if (hp)
        hostname = hp->h_name;
    else
        hostname = "UNKNOWN HOST";

    printf( "recvtcp read %d buffers from %s port %d\n", count, hostname,
            ntohs(sinhim.sin_port) );

    exit(0);
}
```

APPENDIX D: TCP/TP4/TCP TRANSLATOR

```
#
#  Generate the TCP/TP4/TCP translator.
#
#   TJB - Southwest Research Institute
#         San Antonio, Texas
#

ALL     =   TRANd

CFLAGS  =   -c
#CFLAGS =   -c -g -DDEBUG

LDFLAGS     =
#LDFLAGS    =   -g

LIB     =   /Lan/Libs/libiso.a
LIBDIR  =   /Lan/Libs
LIBS    =   $(LIBDIR)/libtp4_if.a \
            $(LIBDIR)/libsystem.a \
            $(LIBDIR)/liblanbuffer.a \
            $(LIBDIR)/liblanutil.a

INCDIR  =   -I/Lan/Src/Lan/Include \
            -I/Lan/Src/Lan/Include/sesstests
INCL    =   $(INCDIR)

#
#  Name of the translator substitute routines.
#

TRANS   =   accept.o    \
            TRAN_utils.o\
            bind.o      \
            close.o     \
            connect.o   \
            debug.o     \
            exit.o      \
            listen.o    \
            gethostnam.o\
            getservent.o\
            getsockopt.o\
            read.o      \
            recv.o      \
            send.o      \
            shutdown.o  \
            socket.o    \
            t_bind.o    \
            t_gethostn.o\
            t_read.o    \
            t_socket.o  \
            t_write.o   \
            transport.o \
            utils.o     \
            write.o

#
#
#

 ll         : $(ALL)

#
#  Translator modules
#
```

```
$(LIB)        :    $(TRANS)
                   rm $(LIB)
                   ar r $(LIB) $(TRANS)
                   ranlib $(LIB)

TRAN_utils.o:      TRAN_utils.c translator.h
                   $(CC) $(CFLAGS)  -o $@ TRAN_utils.c $(INCL)

accept.o      :    accept.c translator.h
                   $(CC) $(CFLAGS)  -o $@ accept.c -DDEBUG $(INCL)

bind.o        :    bind.c translator.h
                   $(CC) $(CFLAGS)  -o $@ bind.c $(INCL)

close.o       :    close.c translator.h
                   $(CC) $(CFLAGS)  -o $@ close.c $(INCL)

connect.o     :    connect.c translator.h
                   $(CC) $(CFLAGS)  -o $@ connect.c -DDEBUG $(INCL)

debug.o       :    debug.c translator.h
                   $(CC) $(CFLAGS)  -o $@ debug.c -DDEBUG $(INCL)

exit.o        :    exit.c translator.h
                   $(CC) $(CFLAGS)  -o $@ exit.c $(INCL)

gethostnam.o:      gethostnam.c translator.h
                   $(CC) $(CFLAGS)  -o $@ gethostnam.c $(INCL)

getservent.o:      getservent.c translator.h
                   $(CC) $(CFLAGS)  -o $@ getservent.c $(INCL)

getsockopt.o:      getsockopt.c translator.h
                   $(CC) $(CFLAGS)  -o $@ getsockopt.c $(INCL)

listen.o      :    listen.c translator.h
                   $(CC) $(CFLAGS)  -o $@ listen.c $(INCL)

read.o        :    read.c translator.h
                   $(CC) $(CFLAGS)  -o $@ read.c $(INCL)

recv.o        :    recv.c translator.h
                   $(CC) $(CFLAGS)  -o $@ recv.c $(INCL)

send.o        :    send.c translator.h
                   $(CC) $(CFLAGS)  -o $@ send.c $(INCL)

shutdown.o    :    shutdown.c translator.h
                   $(CC) $(CFLAGS)  -o $@ shutdown.c $(INCL)

socket.o      :    socket.c translator.h
                   $(CC) $(CFLAGS)  -o $@ socket.c $(INCL)

t_bind.o      :    t_bind.s
                   as -o t_bind.o t_bind.s

t_gethostn.o:      t_gethostn.s
                   as -o t_gethostn.o t_gethostn.s

t_read.o      :    t_read.s
                   as -o t_read.o t_read.s

t_socket.o    :    t_socket.s
```

```
                        as -o t_socket.o t_socket.s

t_write.o      :        t_write.s
                        as -o t_write.o t_write.s

transport.o :           transport.c translator.h
                        $(CC) $(CFLAGS)  -o $@ transport.c $(INCL)

utils.o        :        utils.c translator.h
                        $(CC) $(CFLAGS)  -o $@ utils.c -DDEBUG $(INCL)

write.o        :        write.c translator.h
                        $(CC) $(CFLAGS)  -o $@ write.c $(INCL)

#
#  Translator daemon
#

TRANd.o:    TRANd.c $(LIB)
            $(CC) $(CFLAGS) -o $@ TRANd.c $(INCL)

TRANd:      TRANd.o
            $(CC) $(LDFLAGS) -o $@ TRANd.o $(LIB) $(LIBS) -ljobs
```

```
/*********************************<---->*********************************
 *
 * FILE NAME:    TRAN_utils.c
 *
 *
 * FILE FUNCTION:
 *
 *   TCP/TP4/TCP Translator routines - these utility routines are used
 *   to manage the file descriptor and process structures maintained in
 *   TRANd's shared memory segment.
 *
 *
 * FILE MODULES:
 *
 *   TRAN_add_proc()
 *   TRAN_attach()
 *   TRAN_get_sock()
 *   TRAN_next_fd()
 *   TRAN_test_pfd()
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 *********************************<---->*********************************/

#include "translator.h"
#include <fcntl.h>
#include <errno.h>
#include <sys/param.h>
#include <sys/shm.h>


/*
 *  Translator globals.
 */

TRAN_memory *TRANm;
int         TRANs;          /* translator semaphore id          */

struct sembuf psemop;
struct sembuf vsemop;
```

```
/**********************************<---->**********************************
 *
 * MODULE NAME:  TRAN_add_proc()
 *
 *
 * MODULE FUNCTION:
 *
 *   This routine is used to add a process to the Translator's list
 *   of processes which are using a TCP/TP4 socket.  This routine is
 *   also used to locate the index of an existing process which is
 *   registered in the table.
 *
 *
 * ASSUMPTIONS:
 *
 *   Module assumes a pointer to the Translator shared memory is
 *   initialized within the current process.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
***********************************<---->**********************************/

int  TRAN_add_proc( pid )

    int pid;

{
    register    int i,j;
    Proc_Struct *p;

    /*
     *  First, check for existing process.
     */

    i=0;
    while ( i<MAX_PROCs )
        {
        if ( TRANm->Proc[i].proc == pid )
            return( i );
        else
            i++;
        }

    /*
     *  Process not found in table, find first available process
     *  table entry.  Wait on shared memory semaphore.
     */
    WAIT( TRANs );
    i=0;
    while ((i<MAX_PROCs) && (TRANm->Proc[i].proc))
        i++;

    /*
     *  If we are at the end of the table, there were none available
     *  so return an error code to caller.
     */

    if ( i == MAX_PROCs )
        {
        SIGNAL( TRANs );
        return( ERR );
        }

    /*
     *  There is an available table entry, add the process and init
     *  the file descriptors.  Release the shared memory semaphore.
     */

    p = (Proc_Struct *) &(TRANm->Proc[i]);
    p->proc = pid;
    SIGNAL( TRANs );

    /*
     *  Initialize file descriptors.
     */

    for ( j=0; j<MAX_FDs; j++ )
        p->fd[j] = NONE;

    return( i );
}
```

```
/**********************************<---->**********************************
 *
 * MODULE NAME:  TRAN_attach()
 *
 *
 * MODULE FUNCTION:
 *
 *   This function attaches the Translator's shared memory segment and
 *   the Translator semaphore.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 **********************************<---->**********************************/

TRAN_memory  *TRAN_attach()
{
    static  char *m = NULL;
    int     mid,
            nbytes;

    /*
     *  See if TRAN_attach() has already been called.
     */

    if ( ! m )
        {

        /*
         *  Get address of page boundry past current segment.
         */

        m = (char *) (((unsigned) sbrk(0) +0xfff) & ~0xfff);

        /*
         *  Find memory size rounded up to nearest page size
         */

        nbytes = (btoc(sizeof(TRAN_memory)) * NBPG);

        /*
         *  Create enough space in the data segment.  Return NULL on failure.
         */

        if ( brk(m + nbytes) )
            return( NULL );

        /*
```

```
         *  Return the shared memory id of the existing
         *  shared memory segment. Don't allocate if not present.
         *  Attach to the shared memory segment at the beginning
         *  of the newly created virtual data segment. (Round down
         *  to the nearest page.) Return NULL on failure.
         */

        if ((mid = shmget(TRANm_KEY,nbytes,0777)) != ERR)
            {
            m = (char *) sbrk(0) - nbytes;
            if ((m = (char *) shmat(mid,m,SHM_RND)) == (char *) ERR)
                return( NULL );
            else
                TRANm = (TRAN_memory *) m;
            }
        else
            return( NULL );
        }

    /*
     *  Attach to shared memory semaphore.
     */

    if ((TRANs=semget(TRANs_KEY,1,0777)) == ERR)
        {
        perror("TRAN_attach: semaphore get failed");
        return( NULL );
        }

    return( (TRAN_memory *) m );
}
```

```
/**********************************<---->**********************************
 *
 * MODULE NAME:  TRAN_get_sock()
 *
 *
 * MODULE FUNCTION:
 *
 *   This function allocates a new socket/file descriptor from the
 *   Translator socket/file descriptor table.  The buf_flag is used
 *   to indicate if shared memory buffers will be needed to transfer
 *   data with TP4.
 *
 *
 * ASSUMPTIONS:
 *
 *   Module assumes a pointer to the Translator shared memory is
 *   initialized.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 **********************************<---->**********************************/

int  TRAN_get_sock( type, family, proto, buf_flag )

    int buf_flag,
        family,
        proto,
        type;

{
    int fd,
        i,j;

    /*
     *  Get a dummy socket from the system.
     */

    if ((fd=t_socket(family,type,proto)) == ERR)
        {
        perror("TRAN_get_sock: t_socket() failed");
        errno = EMFILE;
        return( ERR );
        }

    /*
     *  See if there is already a socket at this table
     *  entry.  If so, there is an error.
    */

    if ( P->fd[fd] != NONE )
        {
        errno = 0;
        perror("TRAN_get_sock: table fd is already active");
        return( ERR );
        }

    /*
     *  Obtain exclusive access to shared memory, and
     *  don't release it until active flag is initialized.
     */

    WAIT( TRANs );
    if ((i=TRAN_next_fd()) == ERR)
        {
        SIGNAL( TRANs );
        perror("TRAN_get_sock: file descriptor table full");
        return( ERR );
        }
    TRANm->FD[i].active     = TRUE;
    SIGNAL( TRANs );

    TRANm->FD[i].blocking   = TRUE;
    TRANm->FD[i].bound      = FALSE;
    TRANm->FD[i].connected  = FALSE;
    TRANm->FD[i].data       = FALSE;
    TRANm->FD[i].listen     = TRUE;
    TRANm->FD[i].protocol   = proto;
    TRANm->FD[i].socket     = TRUE;
    TRANm->FD[i].type       = type;
    TRANm->FD[i].use_count  = 1;
    TRANm->FD[i].ev.msg_queue = 0;
    TRANm->FD[i].laddr.sock.sa.sin_family = family;

    if ( buf_flag )
        {
        TRANm->FD[i].rcv_buf = balloc( TP_BUF );
        TRANm->FD[i].snd_buf = balloc( TP_BUF );

        if ((TRANm->FD[i].rcv_buf == NULL) ||
            (TRANm->FD[i].snd_buf == NULL))
            {
            WAIT( TRANs );
            TRANm->FD[i].active     = TRUE;
            SIGNAL( TRANs );
            perror("TRAN_get_sock() - no buffers" );
            return( ERR );
            }
        }

    /*
     *  Initialize process file descriptor pointer to point to
     *  new file descriptor.
     */

    P->fd[fd] = i;

    return( fd );
}
```

```
/**********************************<---->**********************************
 *
 * MODULE NAME:  TRAN_next_fd()
 *
 *
 * MODULE FUNCTION:
 *
 *   Function locates the next available file descriptor in the Translator's
 *   file descriptor table.
 *
 *
 * ASSUMPTIONS:
 *
 *   Module assumes a pointer to the Translator shared memory is
 *   initialized.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
**********************************<---->**********************************/

int  TRAN_next_fd()
{
    register int i=0;


    while ((i<MAX_FDs) && (TRANm->FD[i].active))
        i++;
    if ( i == MAX_FDs )
        {
        errno = EMFILE;
        return( ERR );
        }
    return( i );
}
```

```
/*********************************<---->*********************************
*
* MODULE NAME:  TRAN_test_pfd()
*
*
* MODULE FUNCTION:
*
*   This function determines if the current process has any available
*   process file descriptor pointers.  The process file descriptor
*   pointers are an index into the Translator's file descriptor table.
*
*
* ASSUMPTIONS:
*
*   Module assumes a pointer to the Translator shared memory is
*   initialized.
*
*   Module assumes global var P points to the current process structure.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /Lan/Translator/Specs
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Release 1.0 - 90/09/27
*
*********************************<---->*********************************/

int  TRAN_test_pfd( flag )

    int flag;

{
    register int i;

    if ( flag == PFD_ANY )
        {
        i=0;
        while ( i < MAX_PFDs )
            if ( P->fd[i] == NONE )
                return( NO_ERR );
            else
                i++;

        return( ERR );
        }
}
```

```
/**********************************<---->**********************************
 *
 * FILE NAME:    TRANd.c
 *
 *
 * FILE FUNCTION:
 *
 *   TCP/ISO/TCP daemon process.  TRANd allocates shared memory, inits
 *   the shared memory, creates the server request sockets, and then
 *   waits for requests on the sockets.  The TRANd serves as the inetd
 *   for the TCP/TP4 sockets.
 *
 *
 * FILE MODULES:
 *
 *   check_procs()
 *   check_socks()
 *   create_TRANm()
 *   create_TRANs()
 *   create_socks()
 *   exit_TRANd()
 *   fork_processes()
 *   init_TRANm()
 *   init_signals()
 *   main()
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
**********************************<---->**********************************/


#include "translator.h"
#include <errno.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/shm.h>
#include <sys/param.h>


/*
 *  Function prototypes.
 */

static TRAN_memory  *create_TRANm();
static int           create_TRANs(),
                     create_socks(),
                     init_TRANm();
static void          check_procs(),
                        check_socks(),
                        exit_TRANd(),
                        init_signals();

/*
 *  Globals.
 */

TRAN_memory *TRANm;          /* translator shared memory pointer */
int          TRANs,          /* translator semaphore id          */
             mid,            /* translator shared memory id      */
             rsh_fd,
             rtime_fd,
             sinlen;
char         *mem;

struct sembuf psemop;
struct sembuf vsemop;

struct sockaddr_in  sinme  = { AF_INET };
struct servent      *sp;
```

```
/***********************************<---->***********************************
 *
 * MODULE NAME:  main()
 *
 *
 * MODULE FUNCTION:
 *
 *   This is the main function of TRANd.  All memory, semaphores are created
 *   and initialized.  After the necessary resources are created, this routine
 *   checks the server sockets for connection requests.  If a request is found,
 *   a server is spawned to process the request.  This process also checks
 *   for dead processes and releases their resources if one is found.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 ***********************************<---->***********************************/

int  main( argc, argv )

    char    **argv;
    int     argc;

{
    /*
     *  Create Translator shared memory.
     */

    if ((TRANm = create_TRANm()) == NULL)
        {
        fprintf( stderr, "TRANd: shared memory creation failed\n");
        exit( ERR );
        }

    /*
     *  Create Translator shared memory access semaphore.
     */

    if ((create_TRANs()) == ERR)
        {
        fprintf( stderr, "TRANd: semaphore creation failed\n");
        exit( ERR );
        }

    /*
     *  Setup signals to deallocate memory in the event this
     *  process is terminated.
     */
    init_signals();

    /*
     *  Initialize the ISO translator shared memory.
     */

    if ( init_TRANm() )
        exit_TRANd( 0 );

    /*
     *  Create server sockets.
     */
/*
    if ( create_socks() )
        exit_TRANd( 0 );
*/
    /*
     *  Check to see if any processes have died ungracefully and
     *  remove their process table entries if they have.
     *
     *  Check for activity on "netd" sockets. If connect requests
     *  are found, spawn processes to handle requests.
     */

    while ( TRUE )
        {
        check_procs();
        check_socks();
        sleep( 5 );
        }
}
```

```
/***********************************<---->***********************************
 *
 * MODULE NAME:  check_procs()
 *
 *
 * MODULE FUNCTION:
 *
 *   Check to see if any processes have died ungracefully and
 *   remove their resources if they have.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
************************************<---->***********************************/

static void  check_procs()
{
    register int i,j;
             int index;

    Proc_Struct *p;

    /*
     *  Loop through all process entries and make sure they are active
     *  by sending the NULL kill signal.
     */

    for ( i=0; i<MAX_PROCs; i++ )
        if ( TRANm->Proc[i].proc )
            {
            p = &(TRANm->Proc[i]);
            if ((kill(p->proc,0)) == ERR)
                {

                /*
                 *  Process is no longer active, get exclusive
                 *  access to shared memory and check all of the
                 *  process file descriptors.
                 */

                WAIT( TRANs );

                for ( j=0; j<MAX_PFDs; j++ )
                    {
                    if ( p->fd[j] != NONE )
                        {

                        /*
```

C-2

```
                     *  A file descriptor exists for a dead process.
                     *  Decrement the use count.  If use count == 0,
                     *  remove the file descriptor and the message
                     *  queue.
                     */

                    index = p->fd[j];
                    TRANm->FD[index].use_count--;
                    if (! TRANm->FD[index].use_count )
                        {
                        TRANm->FD[index].active = FALSE;
                        if ( TRANm->FD[index].ev.msg_queue )
                            msgctl( TRANm->FD[index].ev.msg_queue, IPC_RMID, 0 );
                        }
                    }
                p->fd[j] = NONE;
                }

            /*
             *  Remove the process entry in the process table.
             */

            p->proc = 0;

            SIGNAL( TRANs );

            ISOlog("TRANd: removed dead pid\n");
            }
        }
}
```

```
/***********************************<---->***********************************
 *
 * MODULE NAME:  check_socks()
 *
 *
 * MODULE FUNCTION:
 *
 *   This process is currently not implemented because the Translator
 *   currently does not support fork() and exec().  This function should
 *   be implemented should TRANd begin to create the "netd" server sockets.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 ***********************************<---->***********************************/

static void  check_socks()
{
}
```

```
/**********************************<---->**********************************
*
* MODULE NAME:  create_TRANm()
*
*
* MODULE FUNCTION:
*
*   This function creates the shared memory segment used by all Translator
*   processes.  This shared memory segment is where the process table and
*   file descriptor table are maintained.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /Lan/Translator/Specs
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Release 1.0 - 90/09/27
*
**********************************<---->**********************************/

static TRAN_memory  *create_TRANm()
{
    int     nbytes;             /* size of memory to get */
    char    *sbrk(),
            *shmat();
    key_t tran_key = TRANm_KEY;

    /*
     *  Get addr of page boundary beyond current segment
     *      sbrk(0) - gets address of the end of the data segment
     */

    mem = (char *) (((unsigned) sbrk(0) +0xfff) & ~0xfff);

    /*
     *  Find memory size rounded up to nearest page size using
     *  the btoc() macro. (bYTES to PAGE cLICKS)
     */

    nbytes = btoc(sizeof(TRAN_memory)) * NBPG;

    if ( brk(mem + nbytes) )
        {
        fprintf( stderr, "TRANd: brk test failed\n" );
        return (NULL);
        }

    /*
     *  Allocate nbytes worth of system shared memory resources.
     *  Let anybody access them. Return bad status if an error
     *  occurs.
     */
    mid = shmget( tran_key, nbytes, IPC_CREAT | 0777 );
    if ( mid != -1 )
        {
        /*
         *  Attach shared memory to nearest page boundary (round down)
         */

        mem = sbrk(0) - nbytes;
        mem = shmat( mid, mem, SHM_RND );

        /*
         * If an error occurs then return bad
         * status to the calling function.
         */

        if ( (int)mem == -1 )
            return( NULL );
        }
    else
        {
        fprintf( stderr, "TRANd: failed getting shared memory\n" );
        return( NULL );
        }

    return( (TRAN_memory *) mem );
}
```

```
/**********************************<---->**********************************
 *
 * MODULE NAME:  create_TRANs()
 *
 *
 * MODULE FUNCTION:
 *
 *   Create the Translator shared memory exclusive access semaphore.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 **********************************<---->**********************************/

static int  create_TRANs()
{
    if ((TRANs=semget(TRANs_KEY,1,IPC_CREAT | 0777)) == ERR)
        return( ERR );

    /*
     *  Don't allow shared memory access yet.
     */

    semctl( TRANs, SEMNUM, SETVAL, 0 );
}
```

```
/**********************************<---->**********************************
 *
 * MODULE NAME:  create_socks()
 *
 *
 * MODULE FUNCTION:
 *
 *   This routine creates the server side of the sockets used to access
 *   the "r*" (rtime, rsh, etc.) network utilities.  Note, if an error is
 *   detected, log the error but return good status so the shared memory
 *   is not removed from the system.
 *
 *   This function is currently not implemented due to the Translator's
 *   inability to support fork() and exec().
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 **********************************<---->**********************************/

static int  create_socks()
{
    int fd;

    /*
     *  Create rtime() server socket.
     */

    if ((rtime_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        {
        ISOlog("TRANd: error creating rtime() socket\n");
        return( NO_ERR );
        }

    sp = getservbyname("time","tcp");
    sinme.sin_port = sp->s_port;

    if ( bind(rtime_fd, &sinme, sizeof(sinme)) < 0 )
        {
        ISOlog("TRANd: error binding rtime() socket\n");
        return( NO_ERR );
        }

    /*DEBUG*/
    /*  See setsockopt() comments.

    fcntl( rtime_fd, F_SETFL, FNDELAY );
```

```
        */

    setsockopt( rtime_fd, SOL_SOCKET, NO_BLOCK, NULL, 0 );

    /*
     *  Create rsh() server socket.  Update the test for existing
     *  sockets in the "if" statement below when the code for rsh()
     *  is added.
     */

    /*
     *  Wipe out file descriptors to get back to zero.
     */

    for ( fd = getdtablesize(); --fd >= 0; )
        if ((fd != 2) && (fd != rtime_fd))
            close( fd );

    if ((fd=open("/dev/null",O_RDWR)) != 0 )
        {
        ISOlog("TRANd: couldn't get file descriptor zero\n");
        return( NO_ERR );
        }
    else
        close( fd );

    return( NO_ERR );
}
```

```
/*********************************<---->**********************************
 *
 * MODULE NAME:  exit_TRANd()
 *
 *
 * MODULE FUNCTION:
 *
 *   This function first removes any resources held by Translator processes.
 *   The TRANd program is then exited.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 *********************************<---->**********************************/

static void  exit_TRANd( signal )

    int signal;

{
    register int i;

    /*
     *  Remove all Transport-Session message queues held by "sockets".
     */

    for ( i=0; i<MAX_FDs; i++ )
        if ( TRANm->FD[i].ev.msg_queue )
            msgctl( TRANm->FD[i].ev.msg_queue, IPC_RMID, 0 );

    /*
     *  Remove shared memory segment.
     */

    shmdt( mem );
    shmctl( mid, IPC_RMID, 0 );

    /*
     *  Remove semaphore.
     */

    semctl( TRANs, IPC_RMID, 0 );

    /*
     *  Bye!
     */

    _exit( 0 );
```

TRANd.c

```
/**********************************<---->**********************************
 *
 * MODULE NAME:  fork_process()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator fork() routine.  This function is currently not
 *   implemented.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 **********************************<---->**********************************/

int  fork_process()
{
    Proc_Struct *p;
    int         i,j,
                pid,
                ppid;


/*

    if ( (pid=fork()) != 0 )
        return( pid );

     *
     *  Otherwise, we are the new process, lets add ourself to the process
     *  descriptor table and init our fds.
     *


     *
     *  Attach to LAN memory.
     *

    if ( LANmat() == NULL )
        {
        ISOlog("fork() - couldn't attach to LAN memory\n");
        exit( ERR );
        }

     *
     *  Attach to translator shared memory.
     *

    if ((TRANm=TRAN_attach()) == NULL)
    {
    ISOlog("fork() - couldn't attach to TRANd memory\n");
    exit( ERR );
    }

 *
 *  Add this new process to the translator process table.
 *

Pidx = TRAN_add_proc( getpid() );
if ( Pidx == ERR )
    {
    ISOlog("fork() - couldn't add a new process\n");
    exit( ERR );
    }

 *
 *  Get the parent's process ID.
 *

ppid = getppid();

 *
 *  Loop through the translator process table looking for this
 *  process' parent.  When the parent is found, set a pointer to
 *  the parent's process table entry; then loop through all of
 *  the parent's file descriptors and set the child's file
 *  descriptors to the same fd as the parent.
 *

i=0;
while ( i<MAX_PROCs )
    {
    if ( TRANm->Proc[i].proc == ppid )
        {
        p = (Proc_Struct *) &(TRANm->Proc[i]);

        debug("TRANd: found parent %d at %d",ppid,i);
        debug("TRANd: child        %d at %d",getpid(),Pidx);
        debug("TRANd: parent proc is %d", p->proc);
        debug("TRANd: child  proc is %d", TRANm->Proc[Pidx].proc);

        for ( j=0; j<MAX_FDs; j++)
            if ( p->fd[j] != NULL )
                {
                TRANm->Proc[Pidx].fd[j] = p->fd[j];
                debug("TRANd: socket at %d",j);
                if ( p->fd[j] != NULL )
                    debug("TRANd: parent port %d child port %d",
                        p->fd[j]->raddr.sock.sa.sin_port,
                        TRANm->Proc[Pidx].fd[j]->raddr.sock.sa.sin_port);
                }

        TRANm->Proc[Pidx].fd[0] = &(TRANm->FD[2]);
        debug("TRANd: - child protocol %d",TRANm->Proc[Pidx].fd[0]->protocol);

        break;
        }
    else
        i++;
    }

 *
 *  If we didn't find the parent in the translator process table,
         *  abort.
         *

        if ( i == MAX_PROCs )
            {
            debug("fork() - couldn't find parent's proc table entry");
            exit( ERR );
            }

        return( pid );

    */

    }
```

```
/***********************************<---->***********************************
*
* MODULE NAME:  init_TRANm()
*
*
* MODULE FUNCTION:
*
*   This function initializes the Translator shared memory segment by
*   initializing the process descriptor table and the file descriptor
*   table.  This function also inits the port number indicator and
*   releases the shared memory semaphore.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /Lan/Translator/Specs
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Release 1.0 - 90/09/27
*
***********************************<---->***********************************/

static int  init_TRANm()
{
    register int i,j;

    /*
     *  Initialize the process table.
     */

    for ( i=0; i<MAX_PROCs; i++ )
        {
        TRANm->Proc[i].proc = 0;
        for ( j=0; j<MAX_PFDs; j++ )
            TRANm->Proc[i].fd[j] = NONE;
        }

    /*
     *  Initialize the socket descriptor table.
     */

    for ( i=0; i<MAX_FDs; i++ )
        TRANm->FD[i].active = FALSE;

    /*
     *  Initialize the next avaliable port indicator.
     */

    TRANm->next_port = IPPORT_RESERVED + 1;

    /*
     *  Turn loose shared memory access.
     */
```

```
        semctl( TRANs, SEMNUM, SETVAL, 1 );

        return( NO_ERR );
    }
```

```
/**********************************<---->*********************************
*
* MODULE NAME:  init_signals()
*
*
* MODULE FUNCTION:
*
*   This process initializes the Unix signal handler to call exit_TRANd()
*   so the Translator resources are removed when this program exits.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /Lan/Translator/Specs
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Release 1.0 - 90/09/27
*
**********************************<---->*********************************/

static void  init_signals()
{
    signal( SIGHUP,  exit_TRANd );          /* 1  */
    signal( SIGINT,  exit_TRANd );          /* 2  */
    signal( SIGTERM, exit_TRANd );          /* 15 */
}
```

```
/**********************************<---->**********************************
*
* MODULE NAME:  accept()
*
*
* MODULE FUNCTION:
*
*   TCP/TP4/TCP Translator accept() replacement routine.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /Lan/Translator/Specs
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Release 1.0 - 90/09/27
*
**********************************<---->**********************************/

#include "translator.h"
#include <errno.h>
#include <memory.h>


TRAN_memory *TRANm;


int  accept( s, addr, addrlen )

    int *addrlen,
        s;

    struct sockaddr_in *addr;

{
    int          fd,
                 index;
    FD_Struct    *f;
    register struct ev *ev;


    /*
     *  Make sure descriptor is valid.
     */

    if (! check_FD(s) )
        {
        errno = EBADF;
        perror("accept() - Invalid socket descriptor");
        return( ERR );
        }

    f = &( TRANm->FD[(P->fd[s])] );
```

```
     /*
      *  Make sure descriptor is a socket.
      */

     if (! f->socket )
         {
         errno = ENOTSOCK;
         perror("accept() - Descriptor is not a socket");
         return( ERR );
         }

     /*
      *  Make sure this socket is of type STREAM.
      */

     if ( f->type != SOCK_STREAM )
         {
         errno = EOPNOTSUPP;
         perror("accept() - Socket type is not STREAM");
         return( ERR );
         }

     /*
      *  Make sure this socket has been "listen"ed on.
      */

     if ( f->listen != ACTIVE )
         {
         errno = EINVAL;
         perror("accept() - Listen call has not been made on socket");
         return( ERR );
         }

     /*
      *  Make sure addr is in "write" area of caller's address space.
      */

     if ( 0 )
         {
         errno = EFAULT;
         perror("accept() - No write permission in address space");
         return( ERR );
         }

     /*
      *  Make sure a connection indication has been received if
      *  socket is marked non-blocking.
      */

     if ((! f->blocking) && (! f->connected))
         {
         errno = EWOULDBLOCK;
         perror("accept() - Non-blocking socket - no available connection");
         return( ERR );
         }

     /*
      *  If non-blocking and no connection, wait for connection.
      */

#ifdef DEBUG
     debug("accept() - passed checks, waiting for connect");
#endif DEBUG
```

```
     while (! f->connected )
         check_TP4_q( f );

     /*
      *  Issue disconnect so we can deactivate the current TSAP.
      */

     tsap_disconnect( f->machp, f->ev.msg_queue );

     /*
      *  Deactivate the current TSAP so we can build a new one, with all
      *  four parts of the Internet style address.  See connect.c.
      */

     tsap_deactivate( f->machp, f->ev.msg_queue );

     /*
      *  Build new TSAPs.  Add local port to remote address and add remote
      *  port to local address.
      */

     f->connected = FALSE;
     f->activated = FALSE;
     f->raddr.sock.sa.sin_pad.Sin_longs[0] = f->laddr.sock.sa.sin_port;
     f->laddr.sock.sa.sin_pad.Sin_longs[0] = f->raddr.sock.sa.sin_port;

     /*
      *  Rebuild transport interface.  init_ifc() will initialize the
      *  event queue and call TSUadd() to add the TSAP.
      */

     if ( init_ifc(&f->laddr.tsap,&f->ev) == ERR )
         {
         perror( "accept() - couldn't create new TSAP" );
         return( ERR );
         }

     f->rcv_buf->length = 0;
     f->activated = TRUE;

#ifdef DEBUG
    debug("accept: remote addr %d %d %d %d %d",
                          f->raddr.sock.sa.sin_family,
                          f->raddr.sock.sa.sin_port,
                          f->raddr.sock.sa.sin_addr.S_un.S_addr,
                          f->raddr.sock.sa.sin_pad.Sin_longs[0],
                          f->raddr.sock.sa.sin_pad.Sin_longs[1] );

    debug("accept: local  addr %d %d %d %d %d",
                          f->laddr.sock.sa.sin_family,
                          f->laddr.sock.sa.sin_port,
                          f->laddr.sock.sa.sin_addr.S_un.S_addr,
                          f->laddr.sock.sa.sin_pad.Sin_longs[0],
                          f->laddr.sock.sa.sin_pad.Sin_longs[1] );
#endif DEBUG

     /*
      *  Wait for connection again.
      */

#ifdef DEBUG
    debug("accept: waiting for 2nd connection indication");
#endif DEBUG
```

```
    while (! f->connected )
        check_TP4_q( f );

#ifdef DEBUG
    debug("accept: ## Connection established to remote");
#endif DEBUG

    /*
     *  Connection has been established, make sure this process can
     *  handle another socket descriptor.
     */

    if ( TRAN_test_pfd( PFD_ANY ) )
        {
        errno = ENOBUFS;
        perror("accept() - File descriptor table full");
        return( ERR );
        }

    /*
     *  Get a new system socket and initialize a table entry for the new
     *  Translator socket.
     */

    fd = TRAN_get_sock( f->type, f->laddr.sock.sa.sin_family, f->protocol, NO_BUF );
    if ( fd == ERR )
        {
        errno = EMFILE;
        perror( "accept() - Couldn't get socket descriptor" );
        return( ERR );
        }

    /*
     *  Assign old socket values to new socket descriptor.
     */

    index = P->fd[fd];

    TRANm->FD[ index ] = *f;
    f = &( TRANm->FD[ index ] );

    /*
     *  Initialize event queue - pointers in new file descriptor currently
     *  point into old structure, so init all pointers to NULL.  Event
     *  queue in new file descriptor will be clean and will receive next
     *  event.
     */

    QInit(&(f->ev.evavail));
    QInit(&(f->ev.evqueue));
    for (ev = f->ev.events; ev < &(f->ev.events[MAXEVS]); ev++)
       {
        QInit(ev);
        QInsert(ev, &(f->ev.evavail));
        }

#ifdef DEBUG
    debug("accept: assigned new process file descriptor");
#endif DEBUG

    /*
     *  Update input parameters.  If the caller has supplied a structure,
     *  fill it in with the remote socket address.
     */
    if ( addr )
        {
        *addr    = f->raddr.sock.sa;
        *addrlen = 4;
        }

    return( fd );
}
```

bind.c

```
/***********************************<---->***********************************
 *
 * MODULE NAME:  bind()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator bind() replacement routine.
 *
 *   Specify the local half of an Internet association -> local address and
 *   local port.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 ***********************************<---->***********************************/

#include "translator.h"
#include <errno.h>


static struct sockaddr_in sin = { AF_INET };
extern TRAN_memory *TRANm;


int  bind( s, name, namelen )

     int namelen,
         s;
     struct sockaddr_in *name;

{
     int          len;
     FD_Struct    *f;


     /*
      *  Make sure descriptor is valid.  If it's valid, set a pointer
      *  to the file descriptor.
      */

     if (! check_FD(s) )
         {
         errno = EBADF;
         perror("bind() - Invalid socket descriptor");
         return( ERR );
         }

     f = &( TRANm->FD[(P->fd[s])] );
```

```
    /*
     *  Make sure descriptor is a socket.
     */

    if (! f->socket )
        {
        errno = ENOTSOCK;
        perror("bind() - Descriptor is not a socket");
        return( ERR );
        }

    /*
     *  Make sure the socket is not already bound.
     */

    if ( f->bound )
        {
        errno = EINVAL;
        perror("bind() - Socket is already bound");
        return( ERR );
        }

    /*
     *  Add code to check the following error conditions:
     *
     *      name is not available on local machine
     *      name is already in use
     *      name is protected
     *      name is not part of caller's address space
     */

    if ( 0 )
        {
        errno = EADDRINUSE;
        perror("bind() - ");
        return( ERR );
        }

    /*
     *  Build the local NSAP.  Note that the NSAP contains the
     *  ethernet number which is obtained from LAN shared memory.
     */

    f->l_nsap.len       = 8;
    f->l_nsap.addr[0]   = AFI;
    f->l_nsap.addr[7]   = LSAP;

    bcopy(((LAN_memory *)LANm)->eth_addr, &f->l_nsap.addr[1], 6);

    /*
     *  See if the caller supplied a requested local port, if not, bind
     *  the system socket to obtain a local port number.
     */

    if ( name->sin_port == 0 )
        {
#ifdef SYS_PORT
        if ( t_bind(s,&sin,sizeof(sin)) )
            {
            perror("bind: t_bind() failed");
            return( ERR );
            }
        /*
         *  Determine what port the system socket is using.
         */

        if ( getsockname(s,&sin,&len) )
            {
            perror("bind: t_getsockname() failed");
            return( ERR );
            }

        /*
         *  Init the translator fd port number to the system port
         *  number.  This ensures that the translator port numbers
         *  are unique.
         */

        f->laddr.sock.sa.sin_port = sin.sin_port;
#else
        f->laddr.sock.sa.sin_port = TRANm->next_port++;
#endif
        }

    /*
     *  Caller supplied a local port, use it.
     */

    else
        f->laddr.sock.sa.sin_port = name->sin_port;

    /*
     *  Convert the local ethernet number to an Internet number and store it
     *  in the local TSAP.
     */

    ether_to_internet( &f->l_nsap.addr[1], &f->laddr.sock.sa.sin_addr );

    /*
     *  Zero PAD portion of TSAP.
     *
     *  Set size of local TSAP.  TSAP contains a complete sockaddr_in
     *  structure (in.h) as the TSAP id.
     */

    f->laddr.sock.sa.sin_pad.Sin_longs[0] = 0;
    f->laddr.sock.sa.sin_pad.Sin_longs[1] = 0;
    f->laddr.tsap.len = sizeof( struct sockaddr_in );

    /*
     *  Mark socket as bound and TSAP activated.
     */

    f->bound        = TRUE;

#ifdef DEBUG
    debug("bind() - %d %d %d %d %d",
                f->laddr.sock.sa.sin_family,
                f->laddr.sock.sa.sin_port,
                f->laddr.sock.sa.sin_addr.S_un.S_addr,
                f->laddr.sock.sa.sin_pad.Sin_longs[0],
                f->laddr.sock.sa.sin_pad.Sin_longs[1] );
#endif DEBUG

    return( NO_ERR );
}
```

```
/**********************************<---->**********************************
 *
 * MODULE NAME:  close()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator close() replacement routine.  This routine
 *   is named T_close() instead of close(), because the Concurrent file
 *   close.s was not available at this time.  If the Concurrent close.s
 *   file becomes available, it should be implemented similarily to
 *   socket.s.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 **********************************<---->**********************************/

#include "translator.h"
#include <errno.h>


TRAN_memory *TRANm;
int          TRANs;

struct sembuf psemop;
struct sembuf vsemop;


int  T_close( s )

    int s;

{
    FD_Struct   *f;

    /*
     *  Make sure descriptor is valid.
     */

    if (! check_FD(s) )
        {
        errno = EBADF;
        perror("close() - Invalid socket descriptor");
        return( ERR );
        }

    f = &( TRANm->FD[(P->fd[s])] );
    /*
     *  Make sure descriptor is a socket.
     */

    f->socket = TRUE;

    if (! f->socket )
        {
        errno = ENOTSOCK;
        perror("close() - Descriptor is not a socket");
        return( ERR );
        }

    /*
     *  Shut the socket down, remove the system socket also.
     */

    if ( f->connected )
        shutdown( s, 2 );
    close( s );

    /*
     *  Remove the fd.
     */

    p->fd[s] = NONE;

    WAIT( TRANs );
    f->use_count--;
    if (! f->use_count )
        f->active = FALSE;
    SIGNAL( TRANs );
}
```

```
/***********************************<---->***********************************
 *
 * MODULE NAME:  connect()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator connect() replacement routine.
 *
 *   Specify the remote half of an Internet association -> remote address
 *   and remote port.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 ***********************************<---->***********************************/

#include "translator.h"
#include <errno.h>


TRAN_memory *TRANm;


int  connect( s, name, namelen )

    int namelen,
        s;

    struct sockaddr_in *name;

{
    FD_Struct    *f;
    int          i,fd;


    /*
     *  Make sure descriptor is valid.
     */

    if (! check_FD(s) )
        {
        errno = EBADF;
        perror("connect() - Invalid socket descriptor");
        return( ERR );
        }

    f = &( TRANm->FD[(P->fd[s])] );
```

```
/*
 *  Make sure descriptor is a socket.
 */

f->socket = TRUE;

if (! f->socket )
    {
    errno = ENOTSOCK;
    perror("connect() - Descriptor is not a socket");
    return( ERR );
    }

/*
 *  Build remote TSAP using the user supplied Internet
 *  address.
 */

f->raddr.sock.sa = *name;
f->raddr.tsap.len = sizeof( struct sockaddr_in );

/*
 *  If socket is not bound, bind().
 */

if (! f->bound )
    {
    name->sin_port = 0;
    if ( bind(s,name,namelen) == ERR )
        return( ERR );
    }

/*
 *  Make sure this socket has not already been connected.
 */

if ( f->connected )
    {
    errno = EISCONN;
    perror("connect() - Socket already connected");
    return( ERR );
    }

/*
 *  Check the following error conditions:
 *
 *      address is not available on this machine
 *      address family cannot be used with this socket
 *      connection timed out
 *      connection refused
 *      network unreachable
 *      address already in use
 *      name specifies area outside the process address space
 */

if ( 0 )
    {
    errno = 0;
    perror("connect() - ");
    return( ERR );
    }

/*
 *  Make sure a connection indication has been received if
```

```
     *  socket is marked non-blocking.
     */

    if ((! f->blocking) && (! f->connected))
        {
        errno = EWOULDBLOCK;
        perror("connect() - Non-blocking socket - no available connection");
        return( ERR );
        }

    /*
     *  Build remote NSAP.  Convert Internet number to ethernet for NSAP.
     */

    f->r_nsap.len = 8;

    if ( internet_to_ether(&f->raddr.sock.sa.sin_addr,f->r_nsap.addr) == ERR )
        {
        errno = 0;
        perror("connect() - couldn't resolve Internet->Ethernet");
        return( ERR );
        }

    /*
     *  Assign remote port to third part of local tsap.  Do this after bind(),
     *  bind() zeros out the third and fourth parts of the address.
     */

    f->laddr.sock.sa.sin_pad.Sin_longs[0] = f->raddr.sock.sa.sin_port;

    /*
     *  Zero out pad portion of remote TSAP address.
     */

    f->raddr.sock.sa.sin_pad.Sin_longs[0] = 0;
    f->raddr.sock.sa.sin_pad.Sin_longs[1] = 0;

#ifdef DEBUG
    debug("connect() local tsap: %d %d %d %d %d",
                            f->laddr.sock.sa.sin_family,
                            f->laddr.sock.sa.sin_port,
                            f->laddr.sock.sa.sin_addr.S_un.S_addr,
                            f->laddr.sock.sa.sin_pad.Sin_longs[0],
                            f->laddr.sock.sa.sin_pad.Sin_longs[1] );
#endif DEBUG

    if ( init_session(&f->laddr.tsap,&f->ev) == ERR )
        {
        errno = ENOBUFS;
        perror( "connect() - Couldn't init_session()" );
        return( ERR );
        }

    /*
     *  Issue connection request to remote host.
     */

    f->machp = (struct Tmachine *) UCONreq( f->ev.msg_queue, &f->laddr.tsap, &f->r_nsap,
                                    &f->raddr.tsap, (qos_type *) 0, 1, 0 );
    if (! f->machp )
        {
        errno = 0;
        perror("connect() - connection request failed");
        return( ERR );
        }

#ifdef DEBUG
    debug("connect() remote addr %d %d %d %d %d",
                            f->raddr.sock.sa.sin_family,
                            f->raddr.sock.sa.sin_port,
                            f->raddr.sock.sa.sin_addr.S_un.S_addr,
                            f->raddr.sock.sa.sin_pad.Sin_longs[0],
                            f->raddr.sock.sa.sin_pad.Sin_longs[1] );

    debug("connect() local  addr %d %d %d %d %d",
                            f->laddr.sock.sa.sin_family,
                            f->laddr.sock.sa.sin_port,
                            f->laddr.sock.sa.sin_addr.S_un.S_addr,
                            f->laddr.sock.sa.sin_pad.Sin_longs[0],
                            f->laddr.sock.sa.sin_pad.Sin_longs[1] );
#endif DEBUG

    /*
     *  If non-blocking and no connection, wait for response connection.
     */

#ifdef DEBUG
    debug("connect()  waiting for 1rst connect response");
#endif DEBUG

    while (! f->connected )
        check_TP4_q( f );

    while ( f->connected )
        check_TP4_q( f );

#ifdef DEBUG
    debug("connect()  received first connection");
#endif DEBUG

    /*
     *  Build new remote TSAP.  Add our local port as the third part of
     *  the remote's TSAP.
     */

    f->connected = FALSE;
    f->activated = FALSE;
    f->raddr.sock.sa.sin_pad.Sin_longs[0] = f->laddr.sock.sa.sin_port;

    f->rcv_buf->length = 0;

    f->activated = TRUE;

    /*
     *  Reconnect with new remote TSAP.
     */

    f->machp = (struct Tmachine *) UCONreq( f->ev.msg_queue, &f->laddr.tsap, &f->r_nsap,
                                            &f->raddr.tsap, (qos_type *) 0, 1, 0 );
    if (! f->machp )
        {
        errno = 0;
        perror("connect() - connection request failed");
        return( ERR );
        }

#ifdef DEBUG
    debug("connect() new remote addr %d %d %d %d %d",
                                        f->raddr.sock.sa.sin_family,
                                        f->raddr.sock.sa.sin_port,
                                        f->raddr.sock.sa.sin_addr.S_un.S_addr,
                                        f->raddr.sock.sa.sin_pad.Sin_longs[0],
                                        f->raddr.sock.sa.sin_pad.Sin_longs[1] );
#endif DEBUG

    /*
     *  If non-blocking and no connection, wait for connection.
     */

#ifdef DEBUG
    debug("connect: waiting for 2nd accept indication");
#endif DEBUG

    while (! f->connected )
        check_TP4_q( f );

#ifdef DEBUG
    debug("connect: ## Connection established to remote");
#endif DEBUG

    return( NO_ERR );
}
```

```
/*********************************<---->*********************************
*
* MODULE NAME:  debug()
*
*
* MODULE FUNCTION:
*
*   TCP/TP4/TCP Translator routine to write debug messages to a host
*   central logfile.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /Lan/Translator/Specs
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Release 1.0 - 90/09/27
*
*********************************<---->*********************************/

#include <stdio.h>


void  debug( format_str, arg1, arg2, arg3, arg4, arg5, arg6 )

    char    *format_str;    /* printf format string         */
    int     arg1,           /* arg1 value                   */
            arg2,           /* arg2 value                   */
            arg3,           /* arg3 value                   */
            arg4,           /* arg4 value                   */
            arg5,           /* arg5 value                   */
            arg6;           /* arg6 value                   */

{
    FILE    *fp;


#ifdef DEBUG

    /*
     *  Open log file, if this doesn't work, return.
     */

/*
    if ((fp = fopen("/Lan/logfile","a")) == NULL)
*/

    if ((fp = fopen("/dev/null","a")) == NULL)
        return;

    /*
     *  Add new line to log file, close the file.
     */
    fprintf( fp, "%5d: ", getpid() );
    fprintf( fp, format_str, arg1, arg2, arg3, arg4, arg5, arg6 );
    fprintf( fp, "\n" );

    fclose( fp );

#endif DEBUG
}
```

```
/***********************************<---->***********************************
*
* FILE NAME:    events.h
*
*
* FILE FUNCTION:
*
*   This file contains the definition of the various events sent back
*   from TP4 to the Translator.  These definitions are very similar to
*   the event definitions used in the IBM Session layer.  They are
*   put here so the Translator can define multiple event queues for a
*   single process easier.
*
*
* FILE MODULES:
*
*   N/A
*
*
* SPECIFICATION DOCUMENTS:
*
*   /Lan/Translator/Specs
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Release 1.0 - 90/09/27
*
***********************************<---->***********************************/

#include "bufflib.h"
#include "transport.h"

/*
 *  Event constants.
 */

#define MAXEVS      5       /* MUST match define in transport.c      */

#define DATIND      90      /* T_DATA.indication             */
#define DATCON      91      /* T_DATA.confirm                */
#define EXPIND      92      /* T_EXPEDITED_DATA.indication   */
#define REAREQ      93      /* S_SYSTEM_READ.request         */
#define UNIIND      94      /* T_UNIT_DATA.indication        */

/*
 *  Event structures.
 *
 *  T_CONNECT.indication parameters
 */

struct conind {
    tsap_selector rem_tsap_id;  /* remote transport service user */
    nsap_address rem_nsap_addr; /* remote NSAP address */
    qos_type qos;               /* quality-of-service */
    int use_xpd;                /* proposed expedited data usage */
    };

/*
 *  T_CONNECT.confirm parameters
 */

struct concon {
    tsap_selector res_tsap_id;  /* responding TSAP address */
    nsap_address res_nsap_addr; /* responding NSAP address */
    int use_xpd;                /* negotiated expedited data usage */
    qos_type qos;               /* quality-of-service */
    };

/*
 *  T_DATA.indication parameters
 */

struct datind {
    int length;         /* length of TSDU */
    int eotsdu;         /* end of TSDU flag */
    };

/*
 *  T_EXPEDITED_DATA.indication parameters
 */

struct expind {
    struct buf *data;           /* user data buffer */
    };

/*
 *  S_SYSTEM_READ.request parameters
 */

struct sreareq {
    unsigned char *address;     /* address of data */
    int length;                 /* length of data */
    };

/*
 *  T_UNIT_DATA.indication parameters
 */

struct uniind {
    tsap_selector rem_tsap_id;  /* remote transport service user */
    nsap_address rem_nsap_addr; /* remote NSAP address */
    qos_type qos;               /* quality-of-service */
    buf_type data;              /* user data buffer */
    };

/*
 *  Transport event structure.
 */

struct ev {
    struct ev *next;        /* next event on the queue */
    struct ev *prev;        /* previous event on the queue */
    int event;              /* type of event */
    struct Tmachine *tcep;  /* transport connection identifier */
    union {
        struct conind ci;   /* T_CONNECT.indication */
        struct concon cc;   /* T_CONNECT.confirm */
        struct datind dt;   /* T_DATA.indication */
        struct expind xi;   /* T_EXPEDITED_DATA.indication */
        struct sreareq sr;  /* S_SYSTEM_READ.request */
        struct uniind ui;   /* T_UNIT_DATA.indication */
    } un;
};

/*
 *  Transport event queue pointer structure.
 */

struct evq {
    struct ev *first;   /* First event in the queue */
    struct ev *last;    /* Last event in the queue */
    };

typedef struct
    {
    int         msg_queue;
    struct evq  evavail;
    struct evq  evqueue;
    struct ev   events[MAXEVS];
    } Event_struct;
```

```
/***********************************<---->***********************************
 *
 * MODULE NAME:  exit()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/ISO/TCP Translator exit() replacement routine.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 ***********************************<---->***********************************/

#include <errno.h>
#include "translator.h"


struct sembuf psemop;
struct sembuf vsemop;

TRAN_memory *TRANm;
int         TRANs;


void  exit( status )

    int status;

{
    register int i;
    FD_Struct    *f;

    /*
     *  Loop through all connections, closing all open sockets.  Release the
     *  dummy file descriptor also.
     *
     *  Deactivate any active TSAPs.
     */

    for ( i=0; i<MAX_PFDs; i++ )
        {

        /*
         *  If the current fd is a connected or active socket,
         *  remove it from TP4.  Make sure to remove the dummy
         *  socket from the system also.
         *
         *  Remove the TP4-session message queue if one exists.
         */

        if ( check_FD(i) )
            {
            f = &( TRANm->FD[(P->fd[i])] );
            if ( f->active )
                T_close( i );
            }
        }

    /*
     *  Remove the pid from the Translator shared memory.
     */

    WAIT( TRANs );
    P->proc = FALSE;
    SIGNAL( TRANs );

    /*
     *  Exit the current program.  We must use the _exit() routine
     *  because the exit() routine was not yet available from
     *  Concurrent.  If the exit.s file is obtained, the exit()
     *  routine should be renamed and called instead of _exit().
     */

    _exit( status );
}
```

```
/***********************************<---->***********************************
 *
 * FILE NAME:    gethostnam.c
 *
 *
 * FILE FUNCTION:
 *
 *   TCP/TP4/TCP Translator get host info replacement routines.
 *
 *
 * FILE MODULES:
 *
 *   endhostent()
 *   gethostbyaddr()
 *   gethostbyname()
 *   gethostent()
 *   sethostent()
 *   getpeername()
 *   getsockname() - this is implemented as T_getsockname()
 *   sethostent()
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 ***********************************<---->***********************************/

#include <errno.h>
#include <memory.h>
#include "translator.h"

TRAN_memory *TRANm;

FILE    *Ht;                /* Host table file pointer       */
int     Hf,                 /* Host table stayopen flag      */
        Hl,                 /* Host table line number        */
        Ho;                 /* Host table open flag          */
```

```
/**********************************<---->**********************************
*
* MODULE NAME:  endhostent()
*
*
* MODULE FUNCTION:
*
*   TCP/TP4/TCP Translator endhostent() replacement routine.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /Lan/Translator/Specs
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Release 1.0 - 90/09/27
*
***********************************<---->**********************************/

void  endhostent()
{
    if ( Ho )
        fclose( Ht );
    Hl = 0;
    Hf = 0;
    Ho = FALSE;
}
```

```
/***********************************<---->***********************************
 *
 * MODULE NAME:  gethostbyaddr()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator gethostbyaddr() replacement routine.
 *
 *   Query the host table and locate a host entry based on the Internet
 *   address specified by the caller.  Return a pointer to a host struct.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 ***********************************<---->***********************************/

struct hostent  *gethostbyaddr( addr, len, type )

    char    *addr;
    int     len,
            type;

{
    struct hostent *host;

    int found = FALSE;

    /*
     *  Open the host table.
     */

    sethostent( 1 );

    /*
     *  Loop until either the host name is found, or the end
     *  of the host table is found.
     */

    while (! found)
        {
        host = gethostent();
        if ( host == NULL )
            found = TRUE;
        else
            if (! memcmp(addr,host->h_addr,4))
                found = TRUE;
        }
    /*
     *  Close the host table and return the hostent pointer.  Note:
     *  this pointer may be NULL.
     */

    endhostent();
    return( host );
}
```

```
/**********************************<---->**********************************
 *
 * MODULE NAME:  gethostbyname()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator gethostbyname() replacement routine.
 *
 *   Query the host table and locate a host entry based on the hostname
 *   specified by the caller.  Return a pointer to a host struct.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 **********************************<---->**********************************/

struct hostent  *gethostbyname( name )

    char *name;

{
    struct hostent *host;

    int found = FALSE;

    /*
     *  Open the host table.
     */

    sethostent( 1 );

    /*
     *  Loop until either the host name is found, or the end
     *  of the host table is found.
     */

    while (! found)
        {
        host = gethostent();
        if ( host == NULL )
            found = TRUE;
        else
            if (! strcmp(name,host->h_name))
                found = TRUE;

            /*
             *  Check aliases also.
```

```
                */
            }

        /*
         *  Close the host table and return the hostent pointer.  Note:
         *  this pointer may be NULL.
         */

        endhostent();
        return( host );
    }
```

```
/***********************************<---->***********************************
 *
 * MODULE NAME:  gethostent()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator gethostent() replacement routine.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 ***********************************<---->***********************************/

struct hostent  *gethostent()
{
    struct hostent *host;
    char    *asc,
            asc_str[4],
            convert_str[3],
            entry[132],
            *ptr,
            token[132];
    int     cont,
            i=0;

    /*
     *  If the host table is not open, try to open it.
     */

    if (! Ho)
        if ((Ht=fopen(HOST_TABLE,"r")) == NULL)
            return( NULL );
        else
            {
            Ho = TRUE;

            /*
             *  If the file has been previously positioned, reset the position.
             */

            if ( Hl > 0 )
                {
                while ( i < Hl )
                fscanf( Ht, "%*[^\n]", entry );
                }
            }
```

```
/*
 *  Loop until the first valid entry (non-comment) or EOF.
 */

host = NULL;
cont = TRUE;
while ( cont )
    {
    if ( fscanf(Ht,"%s",token) == EOF )
        {
        cont = FALSE;
        break;
        }

    H1++;

    /*
     *  Ignore comment lines.
     */

    if ( token[0] != '#' )
        {
        cont = FALSE;

        /*
         *  Allocate a host entry structure.
         */

        host = (struct hostent *) malloc ( sizeof(struct hostent) );
        if ( host != NULL )
            {
            /*
             *  Allocate address space.
             */
            host->h_addr        = (char *) malloc( 4 );
            host->h_length      = 4;
            host->h_addrtype    = AF_INET;

            /*
             *  Get the first octet of the Internet number.
             */
            asc = asc_str;
            ptr = token;
            while ((*ptr != '.') && (*ptr != NULL))
                {
                *asc = *ptr;
                asc++;
                ptr++;
                }
            *asc = NULL;
            host->h_addr[0] = atoi( asc_str );

            /*
             *  Get the second octet of the Internet number.
             */
            asc = asc_str;
            ptr++;
            while ((*ptr != '.') && (*ptr != NULL))
                {
                *asc = *ptr;
                asc++;
                ptr++;
                }
            *asc = NULL;
            host->h_addr[1] = atoi( asc_str );

            /*
             *  Get the third octet of the Internet number.
             */
            asc = asc_str;
            ptr++;
            while ((*ptr != '.') && (*ptr != NULL))
                {
                *asc = *ptr;
                asc++;
                ptr++;
                }
            *asc = NULL;
            host->h_addr[2] = atoi( asc_str );

            /*
             *  Get the fourth octet of the Internet number.
             */
            ptr++;
            sscanf( ptr, "%s", asc_str );
            host->h_addr[3] = atoi( asc_str );

            /*
             *  Get the ethernet number, store it in a global
             *  and convert its type.
             */

            fscanf( Ht, "%s", Host_Ether_Asc );
            for ( convert_str[2]='\0', i=0; i<6; i++ )
                {
                memcpy( convert_str, &Host_Ether_Asc[i*2], 2 );
                Host_Ether_Int[i] = atoh( convert_str );
                }

            /*
             *  Get the host name.
             */

            fscanf( Ht, "%s", token );
            host->h_name        = (char *) malloc( strlen(token)+1 );
            strcpy( host->h_name, token );

            /*
             *  Get the aliases.
             */

            host->h_aliases     = NULL;

            /*
             *  Get the rest of the line and discard.
             */

            fscanf( Ht, "%*[^\n]", token );
            }
        }
    else
        fscanf( Ht, "%*[^\n]", token );
    }

/*
 *  If the stayopen flag is not set, close the file.
 */
```

```
    if (! Hf)
        {
        fclose( Ht );
        Ho = FALSE;
        }

    return( host );
}
```

```
/**********************************<---->***********************************
 *
 * MODULE NAME:  getpeername()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator getpeername() replacement routine.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 **********************************<---->***********************************/

int  getpeername( s, name, namelen )

    int     namelen,
            s;

    struct sockaddr_in *name;

{
    FD_Struct *f;

    /*
     *  Test the following error conditions:
     *
     *      s is a file, not a descriptor
     *      no buffers
     *      name is bad address
     */

    if ( 0 )
        {
        errno = ENOTSOCK;
        perror("getpeername() - ");
        return( ERR );
        }

    if (! check_FD(s) )
        {
        errno = EBADF;
        perror("getpeername() - Invalid socket descriptor");
        return( ERR );
        }

    f = &( TRANm->FD[(P->fd[s])] );
    if (! f->socket)
        {
        errno = ENOTSOCK;
        perror("getpeername() - Descriptor is not a socket");
        return( ERR );
        }

    /*
     *  Return the remote port.
     */

    *name = f->raddr.sock.sa;

    return( NO_ERR );
}
```

```
/**********************************<---->***********************************
 *
 * MODULE NAME:  T_getsockname()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator getsockname() replacement routine.  The assembly
 *   version of getsockname() was not available, so use T_getsockname()
 *   until the getsockname.s file is obtained from Concurrent.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 **********************************<---->**********************************/

int  T_getsockname( s, name, namelen )

    int      namelen,
             s;

    struct sockaddr_in *name;

{
    FD_Struct *f;

    /*
     *  Test the following error conditions:
     *
     *      s not valid descriptor
     *      s is a file, not a descriptor
     *      no buffers
     *      name is bad address
     */

    if ( 0 )
        {
        errno = ENOTSOCK;
        perror("getsockname() - ");
        return( ERR );
        }

    if (! check_FD(s) )
        {
        errno = EBADF;
        perror("getsockname() - Invalid socket descriptor");
        return( ERR );
        }
    f = &( TRANm->FD[(P->fd[s])] );

    if (! f->socket)
        {
        errno = ENOTSOCK;
        perror("getsockname() - Descriptor is not a socket");
        return( ERR );
        }

    /*
     *  Return local address.
     */

    *name = f->laddr.sock.sa;

    return( NO_ERR );
}
```

```
/***********************************<---->***********************************
 *
 * MODULE NAME:  sethostent()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator sethostent() replacement routine.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 ***********************************<---->***********************************/

int  sethostent( stayopen )

    int stayopen;

{
    /*
     *  Open host table and setup globals: file pointer, stayopen flag,
     *  last line number.
     */

    Ht = fopen(HOST_TABLE,"r");
    Hf = stayopen;
    Hl = 0;
    if ( Ht != NULL )
        Ho = TRUE;
}
```

```
/**********************************<---->***********************************
 *
 * FILE NAME:    getservent.c
 *
 *
 * FILE FUNCTION:
 *
 *   TCP/TP4/TCP Translator "get TCP server" replacement routines.
 *
 *
 * FILE MODULES:
 *
 *   endservent()
 *   getservbyname()
 *   getservbyport()
 *   getservent()
 *   setservent()
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
***********************************<---->**********************************/

#include <errno.h>
#include <memory.h>
#include "translator.h"


FILE    *St;                   /* Server table file pointer        */
int     Sf,                    /* Server table stayopen flag       */
        Sl,                    /* Server table line number         */
        So;                    /* Server table open flag           */
```

```
/***********************************<---->**********************************
 *
 * MODULE NAME:  endservent()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator endservent() replacement routine.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
***********************************<---->**********************************/

void  endservent()
{
    /*
     *  If the server table is open, close it.  Reset flags.
     */

    if ( So )
        fclose( St );
    Sl = 0;
    Sf = 0;
    So = FALSE;
}
```

```
/*********************************<---->*********************************
*
* MODULE NAME:  getservbyname()
*
*
* MODULE FUNCTION:
*
*   TCP/TP4/TCP Translator getservbyname() replacement routine.
*
*   Query the server table and locate a server entry based on the server
*   name specified by the caller.  Return a pointer to a server struct.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /Lan/Translator/Specs
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Release 1.0 - 90/09/27
*
*********************************<---->*********************************/

struct servent  *getservbyname( name, proto )

    char    *name,
            *proto;

{
    struct servent *serv;

    int found = FALSE;

    /*
     *  Open the server table.
     */

    setservent( 1 );

    /*
     *  Loop until either the server name is found, or the end
     *  of the server table is found.
     */

    while (! found)
        {
        serv = getservent();
        if ( serv == NULL )
            found = TRUE;
        else
            if (! strcmp(name,serv->s_name))
                if (! strcmp(proto,serv->s_proto))
                    found = TRUE;
        }
        /*
         *  Close the server table and return the servent pointer.  Note:
         *  this pointer may be NULL.
         */

        endservent();
        return( serv );
    }
```

```
/*********************************<---->**********************************
*
* MODULE NAME:  getservbyport()
*
*
* MODULE FUNCTION:
*
*   TCP/TP4/TCP Translator getservbyport() replacement routine.
*
*   Query the server table and locate a server entry based on the port
*   specified by the caller.  Return a pointer to a server struct.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /Lan/Translator/Specs
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Release 1.0 - 90/09/27
*
*********************************<---->*********************************/

struct servent  *getservbyport( port, proto )

    int     port;
    char    *proto;

{
    struct servent *serv;

    int found = FALSE;

    /*
     *  Open the server table.
     */

    setservent( 1 );

    /*
     *  Loop until either the server port is found, or the end
     *  of the server table is found.
     */

    while (! found)
        {
        serv = getservent();
        if ( serv == NULL )
            found = TRUE;
        else
            if ( port == (int) serv->s_port )
                if (! strcmp(proto,serv->s_proto))
                    found = TRUE;
                /*
                 *  Check aliases also.
                 */
            }

        /*
         *  Close the server table and return the servent pointer.  Note:
         *  this pointer may be NULL.
         */

        endservent();
        return( serv );
    }
```

```
/***********************************<---->**********************************
*
* MODULE NAME:  getservent()
*
*
* MODULE FUNCTION:
*
*   TCP/TP4/TCP Translator getservent() replacement routine.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /Lan/Translator/Specs
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Release 1.0 - 90/09/27
*
***********************************<---->**********************************/

struct servent  *getservent()
{
    struct servent *serv;
    char    token[132];
    int     cont,
            i=0;

    /*
     *  If the server table is not open, try to open it.
     */

    if (! So)
        if ((St=fopen(SERV_TABLE,"r")) == NULL)
            return( NULL );
        else
            {
            So = TRUE;

            /*
             *  If the file has been previously positioned, reset the position.
             */

            if ( Sl > 0 )
                {
                while ( i < Sl )
                fscanf( St, "%*[^\n]", token );
                }
            }

    /*
     *  Loop until the first valid entry (non-comment) or EOF.
     */

    serv = NULL;
```

```
        cont = TRUE;
        while ( cont )
            {
            if ( fscanf(St,"%s",token) == EOF )
                {
                cont = FALSE;
                break;
                }

            Sl++;

            /*
             *  Ignore comment lines.
             */

            if ( token[0] != '#' )
                {
                cont = FALSE;

                /*
                 *  Allocate a server entry structure.
                 */

                serv = (struct servent *) malloc ( sizeof(struct servent) );
                if ( serv != NULL )
                    {

                    /*
                     *  Get the server name.
                     */

                    serv->s_name = (char *) malloc( strlen(token)+1 );
                    strcpy( serv->s_name, token );

                    /*
                     *  Get the server port.
                     */

                    fscanf( St, "%d", &serv->s_port );

                    /*
                     *  Get the server protocol.
                     */

                    fscanf( St, "%c", token );
                    fscanf( St, "%s", token );
                    serv->s_proto = (char *) malloc( strlen(token)+1 );
                    strcpy( serv->s_proto, token );

                    /*
                     *  Get the server aliases.
                     */

                    serv->s_aliases     = NULL;

                    /*
                     *  Get the rest of the line and discard.
                     */

                    fscanf( St, "%*[^\n]", token );
                    }
                }
            else
                fscanf( St, "%*[^\n]", token );
        }

    /*
     *  If the stayopen flag is not set, close the file.
     */

    if (! Sf)
        {
        fclose( St );
        So = FALSE;
        }

    return( serv );
}
```

```
/**********************************<---->**********************************
 *
 * MODULE NAME:  setservent()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator setservent() replacement routine.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 **********************************<---->**********************************/

int  setservent( stayopen )

    int stayopen;

{
    /*
     *  Open server table and setup globals: file pointer, stayopen flag,
     *  last line number.
     */

    St = fopen(SERV_TABLE,"r");
    Sf = stayopen;
    Sl = 0;
    if ( St != NULL )
        So = TRUE;
}
```

```
/**********************************<---->**********************************
*
* FILE NAME:    getsockopt.c
*
*
* FILE FUNCTION:
*
*   TCP/TP4/TCP Translator getsockopt(), setsockopt() replacement routines.
*
*
* FILE MODULES:
*
*   getsockopt() - not implemented currently
*   setsockopt()
*
*
* SPECIFICATION DOCUMENTS:
*
*   /Lan/Translator/Specs
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Release 1.0 - 90/09/27
*
**********************************<---->**********************************/

#include <errno.h>
#include "translator.h"


TRAN_memory *TRANm;
```

```
/**********************************<---->**********************************
 *
 * MODULE NAME:  setsockopt()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator setsockopt() replacement routine.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
**********************************<---->**********************************/

int  setsockopt( s, level, optname, optval, optlen )

    int     level,
            optlen,
            optname,
            s;

    char    *optval;

{
    FD_Struct    *f;
    int          rc = NO_ERR;

    /*
     *  Make sure descriptor is valid.
     */

    if (! check_FD(s) )
        {
        errno = EBADF;
        perror("setsockopt() - Invalid socket descriptor");
        return( ERR );
        }

    f = &( TRANm->FD[(P->fd[s])] );

    /*
     *  Make sure descriptor is a socket.
     */

    if (! f->socket )
        {
        errno = ENOTSOCK;
        perror("setsockopt() - Descriptor is not a socket");
        return( ERR );
        }

    /*
     *  Check the following error conditions:
     *
     *      named option is not in effect
     *      named option is unknown at the specified level
     *      reference to area outside of address space
     */

    if ( 0 )
        {
        errno = EINVAL;
        perror("setsockopt() - ");
        return( ERR );
        }

    /*
     *  Set no-wait, no-block option on this socket.
     */

    switch ( level )
        {
        case SOL_SOCKET :
                    switch ( optname )
                        {
                        case NO_BLOCK :
                                    f->blocking = FALSE;
                                    break;
                        case SO_DEBUG :
                        case SO_DONTLINGER :
                                    break;
                        default :
                                    errno = EINVAL;
                                    perror("setsockopt() - Invalid option specified");
                                    rc = ERR;
                                    break;
                        }
                    break;

        default :
                    errno = EINVAL;
                    perror("setsockopt() - Invalid level specified");
                    rc = ERR;
                    break;
        }

    return( rc );
}
```

```
/************************************<---->************************************
 *
 * MODULE NAME:  listen()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator listen() replacement routine.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 ************************************<---->************************************/

#include <errno.h>
#include "translator.h"


TRAN_memory *TRANm;


int  listen( s, backlog )

    int backlog,
        s;

{
    FD_Struct *f;

    /*
     *  Make sure descriptor is valid.
     */

    if (! check_FD(s) )
        {
        errno = EBADF;
        perror("listen() - Invalid socket descriptor");
        return( ERR );
        }

    f = &( TRANm->FD[(P->fd[s])] );

    /*
     *  Make sure descriptor is a socket.
     */

    if (! f->socket )
        {
        errno = ENOTSOCK;
        perror("listen() - Descriptor is not a socket");
        return( ERR );
        }

    /*
     *  Make sure this type of socket supports a listen() call.
     */

    if (! f->listen )
        {
        errno = EOPNOTSUPP;
        perror("listen() - Socket type does not support listen()");
        return( ERR );
        }
    else
        f->listen = ACTIVE;

#ifdef DEBUG
    debug("listen() -  %d %d %d %d %d",
                f->laddr.sock.sa.sin_family,
                f->laddr.sock.sa.sin_port,
                f->laddr.sock.sa.sin_addr.S_un.S_addr,
                f->laddr.sock.sa.sin_pad.Sin_longs[0],
                f->laddr.sock.sa.sin_pad.Sin_longs[1] );
#endif DEBUG

    /*
     *  Initialize this TSAP.
     */

    if ( init_session(&f->laddr.tsap,&f->ev) == ERR )
        {
        errno = ENOBUFS;
        perror( "listen() - Couldn't init_session()" );
        return( ERR );
        }

    /*
     *  Wait for a connection indication on this VC.
     */

    while (! f->connected )
        check_TP4_q( f );

    return( NO_ERR );
}
```

```
/**********************************<---->**********************************
 *
 * MODULE NAME:  read()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator read() replacement routine.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 **********************************<---->**********************************/

#include "translator.h"
#include <errno.h>


int  read( d, buf, nbytes )

    int         d;
    char        *buf;
    unsigned    nbytes;

{
    /*
     *  See if the system has a socket, if so, then use the
     *  Translator recv() function to read the data from the
     *  TP4 VC.  If the system does not have a socket, then
     *  read the data from the file using the standard
     *  RTU read() function.
     */

    if ( getsockopt(d,SOL_SOCKET,SO_DEBUG,0,0) == ERR )
        if ((errno == EBADF) || (errno == ENOTSOCK))
            return( t_read(d,buf,nbytes) );

    return( recv(d,buf,nbytes,0) );
}
```

PRECEDING PAGE BLANK NOT FILMED

```
/**********************************<---->**********************************
 *
 * FILE NAME:    recv.c
 *
 *
 * FILE FUNCTION:
 *
 *   File contains the TCP/TP4/TCP Translator socket "receive" data routines.
 *
 *
 * FILE MODULES:
 *
 *   recv()
 *   recvfrom()
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 **********************************<---->**********************************/


#include <memory.h>
#include <errno.h>
#include "translator.h"


TRAN_memory *TRANm;
```

```
/**********************************<---->**********************************
 *
 * MODULE NAME:  recv()
 *
 *
 * MODULE FUNCTION:
 *
 *    TCP/TP4/TCP Translator recv() replacement routine.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *    /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *    Timothy J. Barton - Software Engineering Section
 *                        Data Systems Department
 *                        Automation and Data Systems Division
 *                        Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *    Release 1.0 - 90/09/27
 *
 **********************************<---->**********************************/

int  recv( s, buf, len, flags )

    int     flags,
            len,
            s;

    char    *buf;

{
    FD_Struct   *f;
    struct buf  *rcv_buf;
    char        *bptr;
    int         rc,
                tc;

    /*
     *  Make sure descriptor is valid.
     */

    if (! check_FD(s) )
        {
        errno = EBADF;
        perror("recv() - Invalid file/socket descriptor");
        return( ERR );
        }

    f = &( TRANm->FD[(P->fd[s])] );

    /*
     *  Make sure descriptor is a socket.
     */

    if (! f->socket )
        {
```

```
        errno = ENOTSOCK;
        perror("recv() - Descriptor is not a socket");
        return( ERR );
        }

    /*
     *  Check the following error conditions:
     *
     *      receive interrupted by delivery of a signal
     *      data to be received in protected address space
     */

    if ( 0 )
        {
        errno = EFAULT;
        perror("recv() - ");
        return( ERR );
        }

    /*
     *  If the caller requested no data, oh well.
     */

    if (! len)
        return( len );

    /*
     *  See if there is enough data in the receive buffer to satisfy
     *  the current request.  If so, copy the data from the receive
     *  buffer to the caller, and update indicators.
     */

    rcv_buf = f->rcv_buf;
    bptr    = buf;
    rc      = 0;

    if ( len <= rcv_buf->length )
        {
        rc = len;
        memcpy( buf, f->rptr, rc );
        f->rptr += rc;
        rcv_buf->length -= rc;
        errno           =  0;
#ifdef DEBUG
        debug("recv: len %d\n",f->rcv_buf->length);
#endif DEBUG
        return( rc );
        }

    /*
     *  See if there is some data in the receive buffer.
     */

    if ( rcv_buf->length > 0 )
        {
        rc = rcv_buf->length;
        memcpy( bptr, f->rptr, rc );
        rcv_buf->length = 0;
        bptr += rc;
        }

    /*
     *  Make sure we are still connected.  If not, return 0 data.
     */
    if (! f->connected )
        {
        errno = 0;
        return( 0 );
        }

    /*
     *  Issue data receive request.
     */

    if ( UDATrcv(f->ev.msg_queue,f->machp,rcv_buf->addr,TP_BUF,0) )
        {
        errno = EFAULT;
        perror("recv() - UDATrcv failed");
        return( ERR );
        }

    f->rptr = (char *) rcv_buf->addr;

    /*
     *  Make sure data is available if socket is marked as non-blocking.
     */

    if ( 0 ) /* if ((! f->blocking) && (! f->data)) */
        {
        errno = EWOULDBLOCK;
        perror("recv() - Non-blocking socket - no available data");
        return( ERR );
        }

    /*
     *  Wait for data indication.
     */

    while ((! f->data) && (f->connected))
        check_TP4_q( f );

    /*
     *  If a disconnect is received, return 0 to caller to indicate no
     *  more data from socket, otherwise clear data received flag.
     */

    if (! f->connected )
        {
        errno = 0;
        return( rc );
        }
    else
        f->data = FALSE;

    /*
     *  If the request size - number of bytes already loaded is less than
     *  the number of bytes received, then copy to the caller's buffer
     *  only as many as they want.
     */

    if ((len-rc) < rcv_buf->length )
        tc = len-rc;
    else
        tc = rcv_buf->length;

    memcpy( bptr, f->rptr, tc );
    rcv_buf->length -= tc;
    f->rptr += tc;

    rc += tc;

    /*
     *  Return the number of data bytes to the caller.
     */

    errno = 0;
    return( rc );
}
```

```
/**********************************<---->**********************************
 *
 * MODULE NAME:  recvfrom()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator recvfrom() replacement routine.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 **********************************<---->**********************************/

int  recvfrom( s, buf, len, flags, name, namelen )

    int      flags,
             len,
             namelen,
             s;
    char     *buf;
    struct   sockaddr_in *name;

{
    FD_Struct   *f;

    /*
     *  Make sure descriptor is valid.
     */

    if (! check_FD(s) )
        {
        errno = EBADF;
        perror("recvfrom() - Invalid socket descriptor");
        return( ERR );
        }

    f = &( TRANm->FD[(P->fd[s])] );

    /*
     *  Make sure descriptor is a socket.
     */

    if (! f->socket )
        {
        errno = ENOTSOCK;
        perror("recvfrom() - Descriptor is not a socket");
        return( ERR );
        }

    /*
     *  If not connected, connect.
     */

    if (! f->connected )
        s = accept( s, name, &namelen );

    return( recv( s, buf ,len, flags ) );
}
```

```
/*********************************<---->**********************************
 *
 * FILE NAME:    send.c
 *
 *
 * FILE FUNCTION:
 *
 *   File contains the TCP/TP4/TCP Translator socket "send" data routines.
 *
 *
 * FILE MODULES:
 *
 *   send()
 *   sendto()
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
**********************************<---->*********************************/


#include <memory.h>
#include <errno.h>
#include "translator.h"


TRAN_memory *TRANm;
```

```
/**********************************<---->**********************************
*
* MODULE NAME:  send()
*
*
* MODULE FUNCTION:
*
*   TCP/TP4/TCP Translator send() replacement routine.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /Lan/Translator/Specs
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Release 1.0 - 90/09/27
*
**********************************<---->**********************************/

int  send( s, msg, len, flags )

    int     flags,
            len,
            s;

    char    *msg;

{
    FD_Struct   *f;
    struct buf  *snd_buf;
    int         rc;


    /*
     *  Make sure descriptor is valid.
     */

    if (! check_FD(s) )
        {
        errno = EBADF;
        perror("send() - Invalid socket descriptor");
        return( ERR );
        }

    f = &( TRANm->FD[(P->fd[s])] );

    if ( f == NULL )
        {
        perror("send() - file descriptor is NULL");
        return( ERR );
        }

    /*
```

```
         *  Make sure descriptor is a socket.
         */

        if (! f->socket )
            {
            errno = ENOTSOCK;
            perror("send() - Descriptor is not a socket");
            return( ERR );
            }

        /*
         *  Check the following error conditions:
         *
         *      invalid caller address space
         *      atomical message to big for socket
         *      non-blocking socket
         */

        if ( 0 )
            {
            errno = EFAULT;
            perror("send() - ");
            return( ERR );
            }

        /*
         *  If there are more bytes to send than our TP4 buffer will allow,
         *  only send as many as TP4 will handle.
         */

        snd_buf = f->snd_buf;
        if ( len > TP_BUF )
            {
            rc = TP_BUF;
            errno = 0;
            perror("send: len bigger than buffer");
            }
        else
            rc = len;

        /*
         *  Load send buffer with data.
         */

        bcopy( msg, snd_buf->addr, rc );

        if ( UDATreq(f->ev.msg_queue,f->machp,snd_buf->addr,rc,1,0) )
            {
            errno = EFAULT;
            perror("send() - UDATreq failed");
            return( ERR );
            }

        /*
         *  Wait for data confirmation.
         */

        while ((! f->data) && (f->connected))
            check_TP4_q( f );

        /*
         *  If a disconnect is received, return 0 to caller to indicate no
         *  more data from socket, otherwise clear data send flag.
         */
```

```
    if (! f->connected )
        {
        errno = 0;
        return( 0 );
        }
    else
        f->data = FALSE;

    /*
     *  Return the number of data bytes to the caller.
     */

    errno = 0;
    return( rc );
}
```

```
/**********************************<---->***********************************
 *
 * MODULE NAME:  sendto()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator sendto() replacement routine.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 **********************************<---->***********************************/

int  sendto( s, msg, len, flags, name, namelen )

    int     flags,
            len,
            namelen,
            s;
    char    *msg;
    struct sockaddr_in *name;

{
    FD_Struct   *f;


    /*
     *  Make sure descriptor is valid.
     */

    if (! check_FD(s) )
        {
        errno = EBADF;
        perror("sendto() - Invalid socket descriptor");
        return( ERR );
        }

    f = &( TRANm->FD[(P->fd[s])] );

    /*
     *  Make sure descriptor is a socket.
     */

    if (! f->socket )
        {
        errno = ENOTSOCK;
        perror("sendto() - Descriptor is not a socket");
        return( ERR );
        }

    /*
     *  If not connected, connect.
     */

    if (! f->connected )
        connect( s, name, namelen );

    return( send(s,msg,len,flags) );
}
```

```
/***********************************<---->***********************************
 *
 * MODULE NAME:  shutdown()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator shutdown() replacement routine.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 ***********************************<---->***********************************/

#include <errno.h>
#include "translator.h"

TRAN_memory *TRANm;


int  shutdown( s, how )

    int how,
        s;

{
    FD_Struct   *f;
    int         index;


    /*
     *  Make sure descriptor is valid.
     */

    if (! check_FD(s) )
        {
        errno = EBADF;
        perror("shutdown() - Invalid socket descriptor");
        return( ERR );
        }

    f = &( TRANm->FD[(P->fd[s])] );

    /*
     *  Make sure s is a socket.
     */

    if (! f->socket )
        {
```

```
        errno = ENOTSOCK;
        perror("shutdown() - Descriptor is not a socket");
        return( ERR );
        }

    /*
     *  Temporary fix, make sure all data has been sent before disconnecting.
     */

/*DEBUG*/

    if ( 1 )
        sleep(1);

    /*
     *  Free the buffers.
     */

    bfree( f->rcv_buf );
    bfree( f->snd_buf );

    /*
     *  Issue disconnect.  If a message queue was created for this
     *  connection, delete it from the system and clear it.
     */

    tsap_disconnect( f->machp, f->ev.msg_queue );
    tsap_deactivate( &(f->laddr.tsap), f->ev.msg_queue );

    if ( f->ev.msg_queue )
        {
        msgctl( f->ev.msg_queue, IPC_RMID, 0 );
        f->ev.msg_queue = 0;
        }

    /*
     *  Mark socket descriptor as disconnected.
     */

    f->connected = FALSE;
    f->activated = FALSE;
}
```

```
/**********************************<---->**********************************
 *
 * MODULE NAME:  socket()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator socket() replacement routine.
 *
 *
 * ASSUMPTIONS:
 *
 *   The P pointer (process table pointer) is initialized within
 *   socket() and should always point to the current pid process
 *   within the whole process.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
**********************************<---->**********************************/

#include "translator.h"
#include <errno.h>

/*
 *  Translator globals.
 */

TRAN_memory *TRANm = NULL;

int Init = 0;

struct sembuf psemop = { SEMNUM, -1, 0 };
struct sembuf vsemop = { SEMNUM,  1, 0 };


/*
 *  socket()
 */

int  socket( af, type, protocol )

    int af,               /* af - address family                          */
        type,             /* com type - stream, data gram, raw            */
        protocol;         /* com protocol                                 */

{
    int          fd;
    /*
     *  Make sure family is Internet.
     */

    if ( af != AF_INET )
        {
        errno = EAFNOSUPPORT;
        perror( "socket() - Invalid address family: %d", af );
        return( ERR );
        }

    /*
     *  Make sure type is not raw.
     */

    if ( type == SOCK_RAW )
        {
        errno = ESOCKTNOSUPPORT;
        perror( "socket() - Invalid type: %d", type );
        return( ERR );
        }

    /*
     *  Check protocol.
     */

    if ( protocol != 0 )
        {
        errno = EPROTONOSUPPORT;
        perror( "socket() - Invalid protocol: %d", protocol );
        return( ERR );
        }

    /*
     *  Try to attach to LAN shared memory.  Initialize session buffers for
     *  this process.
     */

    if ( ! Init )
        {
        if ( LANmat() == NULL )
            {
            errno = ENOBUFS;
            perror( "socket() - Couldn't get LAN shared memory  ");
            perror( "socket() - Are LANdaemon and TP4 running?  ");
            return( ERR );
            }

        if ( init_buffers(8,TP_BUF) == ERR )
            {
            errno = ENOBUFS;
            perror( "socket() - Couldn't init_buffers()\n" );
            return( ERR );
            }

        /*
         *  Attach to the translator shared memory.
         */

        if ((TRANm=TRAN_attach()) == NULL)
            {
            errno = ENOBUFS;
            perror( "socket() - Couldn't attach to Translator shared memory");
```

```
                return( ERR );
                }

            Init = TRUE;
            }

#ifdef DEBUG
    debug("socket() - attached to LAN memory");
#endif DEBUG

        /*
         *  If the current process is not in the translator process table,
         *  add it.  Save the process table index for this process.
         */

        Pidx = TRAN_add_proc( getpid() );
        if ( Pidx == ERR )
            {
            errno = EMFILE;
            perror( "socket() - No available process structure" );
            return( ERR );
            }

        /*
         *  Make sure there is an available descriptor in the table.  If
         *  the system uses more sockets than our table allows, it will
         *  be detected here.
         */

        P = (Proc_Struct *) &(TRANm->Proc[Pidx]);

        if ( TRAN_test_pfd( PFD_ANY ) )
            {
            errno = EMFILE;
            perror( "socket() - No available file descriptors" );
            return( ERR );
            }

        /*
         *  Get a new system socket and initialize a table entry for the new
         *  Translator socket.
         */

        fd = TRAN_get_sock( type, af, protocol, GET_BUF );
        if ( fd == ERR )
            {
            errno = EMFILE;
            perror( "socket() - Couldn't get socket descriptor" );
            return( ERR );
            }

#ifdef DEBUG
    debug("socket() - created socket %d",fd);
#endif DEBUG

        /*
         *  Return the socket descriptor number.
         */

        return( fd );
}
```

```
/**********************************<---->**********************************
 *
 * FILE NAME:    translator.h
 *
 *
 * FILE FUNCTION:
 *
 *   This file contains the structures and constants used by the
 *   TCP/TP4/TCP Translator.
 *
 *
 * FILE MODULES:
 *
 *   N/A
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 **********************************<---->**********************************/

#include <stdio.h>
#include <ctype.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/socket.h>
#include <sys/utsname.h>
#include <net/in.h>

#include "address.h"
#include "transport.h"
#include "bufflib.h"
#include "Ttypes.h"
#include "Tdefn.h"
#include "session.h"
#include "states.h"
#include "LANshrmem.h"
#include "events.h"

/*
 *  Translator constants.
 */

#ifndef TRUE
#define TRUE        1
#endif

#ifndef FALSE
```

```
#define FALSE       0
#endif

#define ACTIVE      2
#define AFI         0x49
#define ERR         -1
#define GET_BUF     1
#define LSAP        0xA8
#define MAX_FDs     20      /* Max number file descriptors          */
#define MAX_PFDs    10      /* Max num fd per process               */
#define MAX_PROCs   5       /* Max number active trans processes    */
#define MAX_VCs     5       /* Max number active trans VCs          */
#define NONE        -1
#define NO_BLOCK    100
#define NO_ERR      0
#define NO_OPEN     0
#define NO_SOCKET   0
#define NO_BUF      0
#define PFD_ANY     1
#define SYS_FD      0
#define SOC_FD      1
#define TP_BUF      8000
#define TRANm_KEY   0x77100000
#define TRANs_KEY   0x77200000
#define YES_OPEN    1
#define YES_SOCKET  1

#define HOST_TABLE  "/Lan/Config/hosts"
#define SERV_TABLE  "/etc/net/services"

/*
 *  Semaphore processing.
 */

#define SEMNUM  0

#define WAIT(s)         semop( s, &psemop, 1 )
#define SIGNAL(s)       semop( s, &vsemop, 1 )

/*
 *  Check these, they may not be needed due to new flags
 *  in socket structure.
 */

int xpdrcvd;

/*
 *  Typedef's.
 */

typedef unsigned short  IP_port;
typedef int             FD_Index;

/*
 *  File descriptor structure maintained for each translator socket.
 */

typedef struct
    {
    int             active,         /* struct in use ?              */
                    activated,      /* TSAP activated flag          */
                    blocking,       /* Block enable flag            */
                    bound,          /* "Bound" flag                 */
                    connected,      /* "Connected" flag             */
                    data,               /* Data indication from TP4      */
                    domain,
                    listen,             /* Can socket be "listened" on   */
                    protocol,
                    socket,             /* Is descriptor a socket (yes) */
                    type,               /* Socket type - stream, etc.    */
                    use_count;          /* num of users                  */
    nsap_address    l_nsap,             /* Local  host NSAP              */
                    r_nsap;             /* Remote host NSAP - see machp */
    struct buf      *rcv_buf,           /* Data receive buffer           */
                    *snd_buf;           /* Data send    buffer           */
    char            *rptr,              /* Data receive buffer pointer   */
                    *sptr;              /* Data send    buffer pointer   */
    struct Tmachine *machp;             /* remote host machine ptr       */
    Event_struct    ev;
    union
        {
        tsap_selector   tsap;
        struct          ts_addr
            {
            int         len;            /* dummy place holder            */
            struct sockaddr_in sa;      /* Local socket addr INET style */
            } sock;
        } laddr, raddr;
    } FD_Struct;


typedef struct
    {
    int          proc;
    FD_Index     fd[ MAX_PFDs ];
    } Proc_Struct;

/*
 *  Translator shared memory structure.
 */

typedef struct
    {
    Proc_Struct Proc[ MAX_PROCs ];
    FD_Struct   FD  [ MAX_FDs ];
    IP_port     next_port;
    } TRAN_memory;

/*
 *  Translator globals.
 */

Proc_Struct *P;
int         Pidx;

IP_port *Tsap_Port;                         /* ptr to port portion tsap */

char    Host_Ether_Asc[16],                 /* gethostent() - with pad  */
        Host_Ether_Int[6];                  /* gethostent()             */

extern char *LANm;                          /* Ptr to LAN shared memory */
extern int  ISOlogging;                     /* ISOlog active flag       */

/*
 *  Translator function prototypes.
 */

FD_Index    TRAN_get_pfd();
```

```
TRAN_memory *TRAN_attach();

int         TRAN_add_proc(),
            TRAN_next_fd(),
            TRAN_test_pfd(),
            atoh(),
            check_TP4_q();

void        debug();

extern struct buf *balloc();
```

```
/**********************************<---->***********************************
 *
 * FILE NAME:    transport.c
 *
 *
 * FILE FUNCTION:
 *
 *   File contains the TCP/TP4/TCP Translator routines which perform very
 *   Transport level specific functions.
 *
 *   Some of the code in this file has been closely modeled on the IBM Session
 *   interface software.  The functions in this file utilize a message queue
 *   pointer to support multiple VCs in a single process.
 *
 *
 * FILE MODULES:
 *
 *   tsap_activate()
 *   tsap_deactivate()
 *   tsap_disconnect()
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 **********************************<---->***********************************/


#include "translator.h"

/* system include files */
#include <sys/param.h>       /* Unix standard system parm definitions   */
#include <sys/msg.h>         /* Unix standard message definitions       */
#include <signal.h>          /* Unix standard signal definitions        */
#include <fcntl.h>           /* Unix standard file control definitions  */
#include <errno.h>           /* Unix standard errno value definitions   */


/* network include files */
#include "system.h"          /* Network system definitions    */
#include "uio.h"             /* ipc structure definitions     */
#include "session_if.h"      /* Session interface definitions */
#include "config.h"          /* Session configuration            */
#include "manifest.h"        /* Network constant definitions     */
#include "commands.h"        /* Session ipc command definitions  */


extern int tp4_qid;          /* tp4's message qid          */
extern int ses_mem_size;
extern struct tp4msgin rmsg;    /* Request msg to tp4           */
extern char *base_addr;


extern Tconind();   /* T_CONNECT.indication event handler */
extern Tconcon();   /* T_CONNECT.confirm event handler */
extern Tdisind();   /* T_DISCONNECT.indication event handler */
extern Tdatind();   /* T_DATA.indication event handler */
extern Tdatcon();   /* T_DATA.confirm event handler */
extern Texpind();   /* T_EXPEDITED_DATA.indication event handler */
extern Tuniind();   /* T_UNIT_DATA.indication event handler */
extern Tsreareq();  /* S_SYSTEM_READ.request event handler */ /* M002 */
extern Tswrireq();  /* S_SYSTEM_WRITE.request event handler */ /* M002 */


TRAN_memory *TRANm;
```

```
/**********************************<---->**********************************
*
* MODULE NAME:  tsap_activate()
*
*
* MODULE FUNCTION:
*
*   TCP/TP4/TCP Translator TSAP activate function.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /Lan/Translator/Specs
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Release 1.0 - 90/09/27
*
***********************************<---->**********************************/

int  tsap_activate( tsap, msg_queue )

    tsap_selector   *tsap;
    int             msg_queue;

{
    /*
     *  Activate our Transport Service Access Point (TSAP).
     */

    if (TSUadd(tsap, msg_queue, Tconind, Tconcon, Tdisind, Tdatind, Tdatcon,
           Texpind, Tuniind, Tsreareq, Tswrireq) < 0)   /* M002 */
        {
        perror("tsap_activate: could not register TSAP");
        return(-1);
        }
}
```

```
/**********************************<---->**********************************
 *
 * MODULE NAME:  tsap_deactivate()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator tsap deactivate function.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
**********************************<---->**********************************/

int  tsap_deactivate( tsap, msg_queue )

    tsap_selector   *tsap;    /* pointer to TSAP address */
    int             msg_queue;

{
    register struct tp4msgin    *mp = &rmsg;    /* ptr to outgoing message struct */
    register struct Treq        *rp = (struct Treq *)&mp->msg.req;
                                                /* ptr to transport request * /

    /*
     *  Initialize the deactivate message.
     */

    mp->type         = SES_REQ;
    rp->userpid      = getpid();
    rp->qid          = msg_queue;
    rp->loc_tsap_id = *tsap;
    rp->cmd          = TDEACTIVATE;

    /*
     *  Send the message to transport.
     */

    msgsnd(tp4_qid,&rmsg,sizeof(rmsg),0);

    /*
     *  Wait on the return code from transport. Don't
     *  care what the result is.
     */

    (void) tp4_resp(FASTTIMER, msg_queue);
}
```

transport.c

```
/*********************************<---->**********************************
 *
 * MODULE NAME:  tsap_disconnect()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator TSAP disconnect function.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
*********************************<---->*********************************/

int  tsap_disconnect( machp, msg_queue )

    struct Tmachine *machp;
    int             msg_queue;

{
    register struct tp4msgin    *mp = &rmsg;    /* ptr to outgoing message struct */
    register struct Treq        *rp = (struct Treq *)&mp->msg.req;
                                                /* ptr to transport request * /

    /*
     *  Initialize the deactivate message.
     */

    mp->type         = SES_REQ;
    rp->machp        = machp;
    rp->address      = NULL;
    rp->userpid      = getpid();
    rp->qid          = msg_queue;
    rp->reason       = 128;
    rp->cmd          = TDISREQ;

    /*
     *  Send the message to transport.
     */

    msgsnd(tp4_qid,mp,sizeof(rmsg),0);
}
```

```
/**********************************<---->***********************************
*
* FILE NAME:    utils.c
*
*
* FILE FUNCTION:
*
*   TCP/TP4/TCP Translator utilities.
*
*
* FILE MODULES:
*
*   atoh()
*   check_FD()
*   check_TP4_q()
*   ether_to_internet()
*   internet_to_ether()
*
*
* SPECIFICATION DOCUMENTS:
*
*   /Lan/Translator/Specs
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Release 1.0 - 90/09/27
*
**********************************<---->**********************************/

#include <fcntl.h>
#include <errno.h>
#include <memory.h>
#include "translator.h"

/*
 *  Translator globals.
 */

TRAN_memory          *TRANm;

extern struct evq    evavail,
                     evqueue;
extern struct ev     events[MAXEVS];
```

C-3

```
/*****************************************************************
 *                                                               *
 *  Name:  atoh                                                  *
 *                                                               *
 *  Synopsis: int atoh(s)                                        *
 *                                                               *
 *             char *s; string to convert                        *
 *                                                               *
 *  Description: Convert a string containing a hexadecimal       *
 *               number entered by the user into its integer     *
 *               equivalent.                                     *
 *                                                               *
 *  Other Inputs: none.                                          *
 *                                                               *
 *  Outputs:                                                     *
 *     Return Value: decimal equivalent of hexadecimal string    *
 *                                                               *
 *  Interfaces: none.                                            *
 *                                                               *
 *  Resources Used: none.                                        *
 *                                                               *
 *  Limitations: none.                                           *
 *                                                               *
 *  Assumptions: Assumes hex string contains only upper case     *
 *               letters.                                        *
 *                                                               *
 *****************************************************************/

int atoh(s)

    char *s;

{
    register int i,n;

    n = 0;
    for(i=0; (isdigit (s[i])) || (s[i]>='A' && s[i]<='F'); ++i)
        {
        if (isdigit (s[i]))
            n = 16 * n + s[i] - '0';
        else
            n = 16 * n + s[i] - 'A' + 10;
        }
    return( n );
}
```

```
/**********************************<---->**********************************
*
* MODULE NAME:  check_FD()
*
*
* MODULE FUNCTION:
*
*   TCP/TP4/TCP Translator check_FD (check file descriptor) utility.  This
*   function determines whether the integer number passed in is a valid FD
*   for the current process.
*
*   returns:    0 - not a valid fd or error detected
*               1 - there is a file descriptor pointer
*
*
* SPECIFICATION DOCUMENTS:
*
*   /Lan/Translator/Specs
*
*
* ORIGINAL AUTHOR AND IDENTIFICATION:
*
*   Timothy J. Barton - Software Engineering Section
*                       Data Systems Department
*                       Automation and Data Systems Division
*                       Southwest Research Institute
*
*
* REVISION HISTORY:
*
*   Release 1.0 - 90/09/27
*
**********************************<---->**********************************/

int  check_FD( fd )

    int fd;

{
    int         idx;

    /*
     *  Make sure this process is attached to translator shared memory.
     */

    if ( TRANm == NULL )
        {
        if ( LANmat() == NULL )
            {
            perror("check_FD() - not attached to TRANm");
            return( 0 );
            }
        if ((TRANm=TRAN_attach()) == NULL)
            {
            perror("check_FD: Couldn't attach to shared memory");
            return( 0 );
            }
        }

    /*
     *  Make sure this process is in the descriptor table.
     */
    idx = TRAN_add_proc( getpid() );
    if ( idx == ERR )
        {
        perror("check_FD: Couldn't add process");
        return( 0 );
        }

    /*
     *  Set a pointer to this process.
     */

    P = (Proc_Struct *) &(TRANm->Proc[idx]);

    /*
     *  Return a logical which indicates if there is a socket that we are
     *  managing at this FD.
     */

    if ( P->fd[fd] == NONE )
        return( 0 );
    else
        return( 1 );
}
```

```
/*********************************<---->**********************************
 *
 * MODULE NAME:  check_TP4_q()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator check_TP4_queue utility.  This routine checks
 *   a VC's message queue from TP4 to see if any events have been recevied.
 *   If a TP4 event is received, it processes it accordingly.
 *
 *   This routine is modeled on the routine by the same name that is used
 *   within the IBM Session layer.
 *
 *   This function should be moved into the new transport.c file.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
*********************************<---->**********************************/

int  check_TP4_q( fp )

    FD_Struct *fp;

{
    register struct Smachine    *s;
    register struct ev          *ev;
    register struct Tmachine    *mp;
             struct buf         *bp;

    buf_type bf;

    int i;


    /*
     *  Check the session indication message queue for
     *  incoming transport events.
     */

    ck_ind_queue( &(fp->ev) );

    /*
     *  Scan queue of incoming transport layer events.
     */

    while ((ev = fp->ev.evqueue.first) != (struct ev *) &(fp->ev.evqueue))
```

```
{

/*
 *  Determine the event type and process it.
 */

mp = ev->tcep;
switch (ev->event)
    {
    case CONIND:        /* T_CONNECT.indication event */

        debug("Received connect indication");
        debug("Issuing connect response");

        /*
         *  Pull data from connection indication struct.
         */

        fp->raddr.tsap = ev->un.ci.rem_tsap_id;
        fp->r_nsap     = ev->un.ci.rem_nsap_addr;

        /*
         *  Connect indication received, issue connect response.
         */

        UCONres(mp, &ev->un.ci.qos, ev->un.ci.use_xpd, 0);
        fp->machp = mp;
        fp->connected = TRUE;
        break;

    case CONCON:        /* T_CONNECT.confirm event */

        debug("Received connect confirm");

        fp->connected = TRUE;
        break;

    case DATIND:        /* T_DATA.indication event */

        debug("Received data indication length %d", ev->un.dt.length );

        fp->data++;
        fp->rcv_buf->length = ev->un.dt.length;
        break;

    case DISIND:        /* T_DISCONNECT.indication event */

        debug("Received disconnect indication");

        fp->connected = FALSE;
        break;

    case DATCON:        /* T_DATA.confirm event */

        debug("Received data confirm");

        fp->data++;
        break;

    case EXPIND:        /* T_EXPEDITED_DATA.indication event */

        bp = (struct buf *)ev->un.xi.data;
        debug("Received expedited data of length %d", bp->length);
        bdump(bp->addr, bp->length, "\texp data rcvd");
```

```
                bfree(bp);
                xpdrcvd++;
                break;

            default:

                debug("Bad transport event=%d, tcep=%x", ev->event, mp);

                fatal();
                break;
            }

        /* Release event structure */

        QMove(ev, &(fp->ev.evavail));
        }

    return(0);
}
```

```
/**********************************<---->**********************************
 *
 * MODULE NAME:  ether_to_internet()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator utility to convert an ethernet number to an
 *   Internet number.  This routine uses the hostent functions which access
 *   the host table at: /Lan/Config/hosts.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
**********************************<---->**********************************/

int  ether_to_internet( ether, inter )

    char    inter[4],
            ether[6];

{
    struct hostent *host;

    int found = FALSE,
        rc;

    /*
     *  Open the host table.
     */

    sethostent( 1 );

    /*
     *  Loop until either the host name is found, or the end
     *  of the host table is found.
     */

    while (! found)
        {
        host = gethostent();
        if ( host == NULL )
            {
            rc = ERR;
            found = TRUE;
            }
        else
            if (! memcmp(ether,Host_Ether_Int,6))
```

```
                {
                rc = NO_ERR;
                found = TRUE;
                memcpy(inter,host->h_addr,4);
                }
        }

    /*
     *  Close the host table and return the hostent pointer.  Note:
     *  this pointer may be NULL.
     */

    endhostent();
    return( rc );
}
```

```
/*********************************<---->*********************************
 *
 * MODULE NAME:  internet_to_ether()
 *
 *
 * MODULE FUNCTION:
 *
 *   TCP/TP4/TCP Translator utility to convert an Internet number to an
 *   ethernet number.  This routine uses the hostent functions which access
 *   the host table at: /Lan/Config/hosts.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
*********************************<---->*********************************/
int  internet_to_ether( inter, ether )

    char    inter[4],
            ether[6];

{
    struct hostent *host;

    int found = FALSE,
        rc;

    /*
     *  Open the host table.
     */

    sethostent( 1 );

    /*
     *  Loop until either the host name is found, or the end
     *  of the host table is found.
     */

    while (! found)
        {
        host = gethostent();
        if ( host == NULL )
            {
            rc = ERR;
            found = TRUE;
            }
        else
            if (! memcmp(inter,host->h_addr,4))
```

```
                {
                rc              = NO_ERR;
                found           = TRUE;
                ether[0]        = AFI;
                ether[7]        = LSAP;
                memcpy(&ether[1],Host_Ether_Int,6);
                }
        }

    /*
     *  Close the host table and return the hostent pointer.  Note:
     *  this pointer may be NULL.
     */

    endhostent();
    return( rc );
}
```

write.c

```
/***********************************<---->**********************************
 *
 * MODULE NAME:  write()
 *
 *
 * MODULE FUNCTION:
 *
 *   File contains the TCP/TP4/TCP Translator socket "write" data
 *   replacement routine.
 *
 *
 * SPECIFICATION DOCUMENTS:
 *
 *   /Lan/Translator/Specs
 *
 *
 * ORIGINAL AUTHOR AND IDENTIFICATION:
 *
 *   Timothy J. Barton - Software Engineering Section
 *                       Data Systems Department
 *                       Automation and Data Systems Division
 *                       Southwest Research Institute
 *
 *
 * REVISION HISTORY:
 *
 *   Release 1.0 - 90/09/27
 *
 ***********************************<---->**********************************/

#include "translator.h"
#include <errno.h>


int  write( filedes, buf, nbyte )

    int         filedes;
    char        *buf;
    unsigned    nbyte;

{
    /*
     *  See if the system has a socket fd, if so, use the Translator
     *  send() routine to write the data across the TP4 VC.  If the
     *  system doesn't have a socket, then write the data using the
     *  Translator write() replacement.
     */

    if ( getsockopt(filedes,SOL_SOCKET,SO_DEBUG,0,0) == ERR )
        if ((errno == EBADF) || (errno == ENOTSOCK))
            return( t_write(filedes,buf,nbyte) );

    return( send(filedes,buf,nbyte,0) );
}
```